jQuery API Documentation

This PDF was generated by jqueryapidoc.andrewbarfield.com on
26 April 2012 from the Raw XML API Dump.

Ajax

Global Ajax Event Handlers

ajaxComplete(handler(event, XMLHttpRequest, ajaxOptions))

Register a handler to be called when Ajax requests complete. This is an Ajax Event.

Arguments

handler(event, XMLHttpRequest, ajaxOptions) - The function to be invoked.

Whenever an Ajax request completes, jQuery triggers the `ajaxComplete` event. Any and all handlers that have been registered with the `.ajaxComplete()` method are executed at this time.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxComplete(function() {
 $(this).text('Triggered ajaxComplete handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request completes, the log message is displayed.

Note: Because `.ajaxComplete()` is implemented as a method of jQuery object instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxComplete` handlers are invoked, regardless of what Ajax request was completed. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxComplete` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. For example, we can restrict our callback to only handling events dealing with a particular URL:

Note: You can get the returned ajax contents by looking at `xhr.responseXML` or `xhr.responseHTML` for xml and html respectively.

```
$('.log').ajaxComplete(function(e, xhr, settings) {
 if (settings.url == 'ajax/test.html') {
  $(this).text('Triggered ajaxComplete handler. The result is ' +
           xhr.responseHTML);
 }
});
```

Example

Show a message when an Ajax request completes.

```
$("#msg").ajaxComplete(function(event,request, settings){
  $(this).append("<li>Request Complete.</li>");
 });
```

ajaxSuccess(handler(event, XMLHttpRequest, ajaxOptions))

Attach a function to be executed whenever an Ajax request completes successfully. This is an Ajax Event.

Arguments

handler(event, XMLHttpRequest, ajaxOptions) - The function to be invoked.

Whenever an Ajax request completes successfully, jQuery triggers the `ajaxSuccess` event. Any and all handlers that have been registered with the `.ajaxSuccess()` method are executed at this time.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxSuccess(function() {
 $(this).text('Triggered ajaxSuccess handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request completes successfully, the log message is displayed.

Note: Because `.ajaxSuccess()` is implemented as a method of jQuery object instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxSuccess` handlers are invoked, regardless of what Ajax request was completed. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxSuccess` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. For example, we can restrict our callback to only handling events dealing with a particular URL:

Note: You can get the returned ajax contents by looking at `xhr.responseXML` or `xhr.responseText` for xml and html respectively.

```
$('.log').ajaxSuccess(function(e, xhr, settings) {
 if (settings.url == 'ajax/test.html') {
  $(this).text('Triggered ajaxSuccess handler. The ajax response was:'
           + xhr.responseText );
 }
});
```

Example

Show a message when an Ajax request completes successfully.

```
$("#msg").ajaxSuccess(function(evt, request, settings){
   $(this).append("<li>Successful Request!</li>");
   });
```

ajaxStop(handler())

Register a handler to be called when all Ajax requests have completed. This is an Ajax Event.

Arguments

handler() - The function to be invoked.

Whenever an Ajax request completes, jQuery checks whether there are any other outstanding Ajax requests. If none remain, jQuery triggers the `ajaxStop` event. Any and all handlers that have been registered with the `.ajaxStop()` method are executed at this time. The `ajaxStop` event is also triggered if the last outstanding Ajax request is cancelled by returning false within the `beforeSend` callback function.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxStop(function() {
 $(this).text('Triggered ajaxStop handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request completes, the log message is displayed.

Because `.ajaxStop()` is implemented as a method of jQuery object instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

Example
Hide a loading message after all the Ajax requests have stopped.

```
$("#loading").ajaxStop(function(){
   $(this).hide();
});
```

ajaxStart(handler())
Register a handler to be called when the first Ajax request begins. This is an Ajax Event.

Arguments
handler() - The function to be invoked.

Whenever an Ajax request is about to be sent, jQuery checks whether there are any other outstanding Ajax requests. If none are in progress, jQuery triggers the `ajaxStart` event. Any and all handlers that have been registered with the `.ajaxStart()` method are executed at this time.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxStart(function() {
 $(this).text('Triggered ajaxStart handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request is sent, the log message is displayed.

Note: Because `.ajaxStart()` is implemented as a method of jQuery object instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

Example
Show a loading message whenever an Ajax request starts (and none is already active).

```
$("#loading").ajaxStart(function(){
  $(this).show();
 });
```

ajaxSend(handler(event, jqXHR, ajaxOptions))
Attach a function to be executed before an Ajax request is sent. This is an Ajax Event.

Arguments
handler(event, jqXHR, ajaxOptions) - The function to be invoked.

Whenever an Ajax request is about to be sent, jQuery triggers the `ajaxSend` event. Any and all handlers that have been registered with the `.ajaxSend()` method are executed at this time.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxSend(function() {
 $(this).text('Triggered ajaxSend handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request is about to begin, the log message is displayed.

Note: Because `.ajaxSend()` is implemented as a method of jQuery instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxSend` handlers are invoked, regardless of what Ajax request is to be sent. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxSend` handler is executed, it is passed the event object, the `jqXHR` object (in version 1.4, `XMLHttpRequest`object), and the settings object that was used in the creation of the Ajax request. For example, we can restrict our callback to only handling events dealing with a particular URL:

```
$('.log').ajaxSend(function(e, jqxhr, settings) {
 if (settings.url == 'ajax/test.html') {
  $(this).text('Triggered ajaxSend handler.');
 }
});
```

Example
Show a message before an Ajax request is sent.

```
$("#msg").ajaxSend(function(evt, request, settings){
    $(this).append("<li>Starting request at " + settings.url + "</li>");
   });
```

ajaxError(handler(event, jqXHR, ajaxSettings, thrownError))
Register a handler to be called when Ajax requests complete with an error. This is an Ajax Event.

Arguments

handler(event, jqXHR, ajaxSettings, thrownError) - The function to be invoked.

Whenever an Ajax request completes with an error, jQuery triggers the `ajaxError` event. Any and all handlers that have been registered with the `.ajaxError()` method are executed at this time.

To observe this method in action, set up a basic Ajax load request.

```
<button class="trigger">Trigger</button>
<div class="result"></div>
<div class="log"></div>
```

Attach the event handler to any element:

```
$("div.log").ajaxError(function() {
 $(this).text( "Triggered ajaxError handler." );
});
```

Now, make an Ajax request using any jQuery method:

```
$("button.trigger").click(function() {
 $("div.result").load( "ajax/missing.html" );
});
```

When the user clicks the button and the Ajax request fails, because the requested file is missing, the log message is displayed.

Note: Because `.ajaxError()` is implemented as a method of jQuery object instances, you can use the `this` keyword within the callback function to refer to the selected elements.

All `ajaxError` handlers are invoked, regardless of what Ajax request was completed. To differentiate between the requests, you can use the parameters passed to the handler. Each time an `ajaxError` handler is executed, it is passed the event object, the `jqXHR` object (prior to jQuery 1.5, the XHR object), and the settings object that was used in the creation of the request. If the request failed because JavaScript raised an exception, the exception object is passed to the handler as a fourth parameter. For example, to restrict the error callback to only handling events dealing with a particular URL:

```
$( "div.log" ).ajaxError(function(e, jqxhr, settings, exception) {
 if ( settings.url == "ajax/missing.html" ) {
  $(this).text( "Triggered ajaxError handler." );
 }
});
```

Example

Show a message when an Ajax request fails.

```
$("#msg").ajaxError(function(event, request, settings){
 $(this).append("<li>Error requesting page " + settings.url + "</li>");
});
```

Helper Functions

serializeArray()

Encode a set of form elements as an array of names and values.

The `.serializeArray()` method creates a JavaScript array of objects, ready to be encoded as a JSON string. It operates on a jQuery object representing a set of form elements. The form elements can be of several types:

```
<form>
 <div><input type="text" name="a" value="1" id="a" /></div>
 <div><input type="text" name="b" value="2" id="b" /></div>
 <div><input type="hidden" name="c" value="3" id="c" /></div>
 <div>
```

```
  <textarea name="d" rows="8" cols="40">4</textarea>
 </div>
 <div><select name="e">
  <option value="5" selected="selected">5</option>
  <option value="6">6</option>
  <option value="7">7</option>
 </select></div>
 <div>
  <input type="checkbox" name="f" value="8" id="f" />
 </div>
 <div>
  <input type="submit" name="g" value="Submit" id="g" />
 </div>
</form>
```

The `.serializeArray()` method uses the standard W3C rules for successful controls to determine which elements it should include; in particular the element cannot be disabled and must contain a `name` attribute. No submit button value is serialized since the form was not submitted using a button. Data from file select elements is not serialized.

This method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization:

```
$('form').submit(function() {
 console.log($(this).serializeArray());
 return false;
});
```

This produces the following data structure (provided that the browser supports `console.log`):

```
[
 {
  name: "a",
  value: "1"
 },
 {
  name: "b",
  value: "2"
 },
 {
  name: "c",
  value: "3"
 },
 {
  name: "d",
  value: "4"
 },
 {
  name: "e",
  value: "5"
 }
]
```

Example

Get the values from a form, iterate through them, and append them to a results display.

```
function showValues() {
   var fields = $(":input").serializeArray();
   $("#results").empty();
   jQuery.each(fields, function(i, field){
    $("#results").append(field.value + " ");
   });
  }
```

```
  $(":checkbox, :radio").click(showValues);
  $("select").change(showValues);
  showValues();
```

serialize()

Encode a set of form elements as a string for submission.

The `.serialize()` method creates a text string in standard URL-encoded notation. It operates on a jQuery object representing a set of form elements. The form elements can be of several types:

```
<form>
 <div><input type="text" name="a" value="1" id="a" /></div>
 <div><input type="text" name="b" value="2" id="b" /></div>
 <div><input type="hidden" name="c" value="3" id="c" /></div>
 <div>
  <textarea name="d" rows="8" cols="40">4</textarea>
 </div>
 <div><select name="e">
  <option value="5" selected="selected">5</option>
  <option value="6">6</option>
  <option value="7">7</option>
 </select></div>
 <div>
  <input type="checkbox" name="f" value="8" id="f" />
 </div>
 <div>
  <input type="submit" name="g" value="Submit" id="g" />
 </div>
</form>
```

The `.serialize()` method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization:

```
$('form').submit(function() {
 alert($(this).serialize());
 return false;
});
```

This produces a standard-looking query string:

```
a=1&b=2&c=3&d=4&e=5
```

Warning: selecting both the form and its children will cause duplicates in the serialized string.

Note: Only "successful controls" are serialized to the string. No submit button value is serialized since the form was not submitted using a button. For a form element's value to be included in the serialized string, the element must have a `name` attribute. Values from checkboxes and radio buttons (`inputs` of type "radio" or "checkbox") are included only if they are checked. Data from file select elements is not serialized.

Example

Serialize a form to a query string, that could be sent to a server in an Ajax request.

```
function showValues() {
   var str = $("form").serialize();
   $("#results").text(str);
  }
  $(":checkbox, :radio").click(showValues);
  $("select").change(showValues);
  showValues();
```

jQuery.param(obj)

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

Arguments

obj - An array or object to serialize.

This function is used internally to convert form element values into a serialized string representation (See .serialize() for more information).

As of jQuery 1.3, the return value of a function is used instead of the function as a String.

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. You can disable this functionality globally by setting `jQuery.ajaxSettings.traditional = true;`.

If the object passed is in an Array, it must be an array of objects in the format returned by .serializeArray()

```
[{name:"first",value:"Rick"},
{name:"last",value:"Astley"},
{name:"job",value:"Rock Star"}]
```

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

Note: Because there is no universally agreed-upon specification for param strings, it is not possible to encode complex data structures using this method in a manner that works ideally across all languages supporting such input. Until such time that there is, the `$.param` method will remain in its current form.

In jQuery 1.4, HTML5 input elements are also serialized.

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));

alert(recursiveEncoded);
alert(recursiveDecoded);
```

The values of `recursiveEncoded` and `recursiveDecoded` are alerted 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);

alert(shallowEncoded);
```

```
alert(shallowDecoded);
```

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

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

Example
Serialize a key/value object.

```
var params = { width:1680, height:1050 };
  var str = jQuery.param(params);
  $("#results").text(str);
```

Example
Serialize a few complex objects

```
// <=1.3.2:
$.param({ a: [2,3,4] }) // "a=2&a=3&a=4"
// >=1.4:
$.param({ a: [2,3,4] }) // "a[]=2&a[]=3&a[]=4"

// <=1.3.2:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a=[object+Object]&d=3&d=4&d=[object+Object]"
// >=1.4:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a[b]=1&a[c]=2&d[]=3&d[]=4&d[2][e]=5"
```

Low-Level Interface

jQuery.ajaxPrefilter([dataTypes], handler(options, originalOptions, jqXHR))
Handle custom Ajax options or modify existing options before each request is sent and before they are processed by `$.ajax()`.

Arguments
dataTypes - An optional string containing one or more space-separated dataTypes
handler(options, originalOptions, jqXHR) - A handler to set default values for future Ajax requests.

A typical prefilter registration using `$.ajaxPrefilter()` looks like this:

```
$.ajaxPrefilter( function( options, originalOptions, jqXHR ) {
 // Modify options, control originalOptions, store jqXHR, etc
});
```

where:
- `options` are the request options
- `originalOptions` are the options as provided to the ajax method, unmodified and, thus, without defaults from `ajaxSettings`
- `jqXHR` is the jqXHR object of the request

Prefilters are a perfect fit when custom options need to be handled. Given the following code, for example, a call to `$.ajax()` would automatically abort a request to the same URL if the custom `abortOnRetry` option is set to `true`:

```
var currentRequests = {};

$.ajaxPrefilter(function( options, originalOptions, jqXHR ) {
 if ( options.abortOnRetry ) {
  if ( currentRequests[ options.url ] ) {
   currentRequests[ options.url ].abort();
  }
  currentRequests[ options.url ] = jqXHR;
```

```
 }
});
```

Prefilters can also be used to modify existing options. For example, the following proxies cross-domain requests through http://mydomain.net/proxy/:

```
$.ajaxPrefilter( function( options ) {
 if ( options.crossDomain ) {
  options.url = "http://mydomain.net/proxy/" + encodeURIComponent( options.url );
  options.crossDomain = false;
 }
});
```

If the optional `dataTypes` argument is supplied, the prefilter will be only be applied to requests with the indicated dataTypes. For example, the following only applies the given prefilter to JSON and script requests:

```
$.ajaxPrefilter( "json script", function( options, originalOptions, jqXHR ) {
 // Modify options, control originalOptions, store jqXHR, etc
});
```

The `$.ajaxPrefilter()` method can also redirect a request to another dataType by returning that dataType. For example, the following sets a request as "script" if the URL has some specific properties defined in a custom `isActuallyScript()` function:

```
$.ajaxPrefilter(function( options ) {
 if ( isActuallyScript( options.url ) ) {
  return "script";
 }
});
```

This would ensure not only that the request is considered "script" but also that all the prefilters specifically attached to the script dataType would be applied to it.

jQuery.ajaxSetup(options)

Set default values for future Ajax requests.

Arguments

options - A set of key/value pairs that configure the default Ajax request. All options are optional.

For details on the settings available for `$.ajaxSetup()`, see `$.ajax()`.

All subsequent Ajax calls using any function will use the new settings, unless overridden by the individual calls, until the next invocation of `$.ajaxSetup()`.

For example, the following sets a default for the `url` parameter before pinging the server repeatedly:

```
$.ajaxSetup({
 url: 'ping.php'
});
```

Now each time an Ajax request is made, the "ping.php" URL will be used automatically:

```
$.ajax({
 // url not set here; uses ping.php
 data: {'name': 'Dan'}
});
```

Note: Global callback functions should be set with their respective global Ajax event handler methods-`.ajaxStart()`, `.ajaxStop()`, `.ajaxComplete()`, `.ajaxError()`, `.ajaxSuccess()`, `.ajaxSend()`-rather than within the `options` object for `$.ajaxSetup()`.

Example

Sets the defaults for Ajax requests to the url "/xmlhttp/", disables global handlers and uses POST instead of GET. The following Ajax requests then sends some data without having to set anything else.

```
$.ajaxSetup({
  url: "/xmlhttp/",
  global: false,
  type: "POST"

 });
 $.ajax({ data: myData });
```

jQuery.ajax(url, [settings])

Perform an asynchronous HTTP (Ajax) request.

Arguments

url - A string containing the URL to which the request is sent.
settings - A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup(). See jQuery.ajax(settings) below for a complete list of all settings.

The `$.ajax()` function underlies all Ajax requests sent by jQuery. It is often unnecessary to directly call this function, as several higher-level alternatives like `$.get()` and `.load()` are available and are easier to use. If less common options are required, though, `$.ajax()` can be used more flexibly.

At its simplest, the `$.ajax()` function can be called with no arguments:

```
$.ajax();
```

Note: Default settings can be set globally by using the `$.ajaxSetup()` function.

This example, using no options, loads the contents of the current page, but does nothing with the result. To use the result, we can implement one of the callback functions.

The jqXHR Object

The jQuery XMLHttpRequest (jqXHR) object returned by `$.ajax()`**as of jQuery 1.5** is a superset of the browser's native XMLHttpRequest object. For example, it contains `responseText` and `responseXML` properties, as well as a `getResponseHeader()` method. When the transport mechanism is something other than XMLHttpRequest (for example, a script tag for a JSONP request) the `jqXHR` object simulates native XHR functionality where possible.

As of jQuery 1.5.1, the `jqXHR` object also contains the `overrideMimeType()` method (it was available in jQuery 1.4.x, as well, but was temporarily removed in jQuery 1.5). The `.overrideMimeType()` method may be used in the `beforeSend()` callback function, for example, to modify the response content-type header:

```
$.ajax({
 url: "http://fiddle.jshell.net/favicon.png",
 beforeSend: function ( xhr ) {
  xhr.overrideMimeType("text/plain; charset=x-user-defined");
 }
}).done(function ( data ) {
 if( console && console.log ) {
  console.log("Sample of data:", data.slice(0, 100));
 }
});
```

The jqXHR objects returned by `$.ajax()` as of jQuery 1.5 implement the Promise interface, giving them all the properties, methods, and behavior of

a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, jqXHR also provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the `$.ajax()` request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback. This allows you to assign multiple callbacks on a single request, and even to assign callbacks after the request may have completed. (If the request is already complete, the callback is fired immediately.)

Deprecation Notice: The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callbacks will be deprecated in jQuery 1.8. To prepare your code for their eventual removal, use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.ajax( "example.php" )
  .done(function() { alert("success"); })
  .fail(function() { alert("error"); })
  .always(function() { alert("complete"); });

// perform other work here ...

// Set another completion function for the request above
jqxhr.always(function() { alert("second complete"); });
```

For backward compatibility with `XMLHttpRequest`, a `jqXHR` object will expose the following properties and methods:

- `readyState`
- `status`
- `statusText`
- `responseXML` and/or `responseText` when the underlying request responded with xml and/or text, respectively
- `setRequestHeader(name, value)` which departs from the standard by replacing the old value with the new one rather than concatenating the new value to the old one
- `getAllResponseHeaders()`
- `getResponseHeader()`
- `abort()`

No `onreadystatechange` mechanism is provided, however, since `success`, `error`, `complete` and `statusCode` cover all conceivable requirements.

Callback Function Queues

The `beforeSend`, `error`, `dataFilter`, `success` and `complete` options all accept callback functions that are invoked at the appropriate times.

As of jQuery 1.5, the `error` (`fail`), `success` (`done`), and `complete` (`always`, as of jQuery 1.6) callback hooks are first-in, first-out managed queues. This means you can assign more than one callback for each hook. See Deferred object methods, which are implemented internally for these `$.ajax()` callback hooks.

The `this` reference within all callbacks is the object in the `context` option passed to `$.ajax` in the settings; if `context` is not specified, `this` is a reference to the Ajax settings themselves.

Some types of Ajax requests, such as JSONP and cross-domain GET requests, do not use XHR; in those cases the `XMLHttpRequest` and `textStatus` parameters passed to the callback are `undefined`.

Here are the callback hooks provided by `$.ajax()`:

- `beforeSend` callback is invoked; it receives the `jqXHR` object and the `settings` map as parameters.
- `error` callbacks are invoked, in the order they are registered, if the request fails. They receive the `jqXHR`, a string indicating the error type, and an exception object if applicable. Some built-in errors will provide a string as the exception object: "abort", "timeout", "No Transport".
- `dataFilter` callback is invoked immediately upon successful receipt of response data. It receives the returned data and the value of `dataType`, and must return the (possibly altered) data to pass on to `success`.
- `success` callbacks are then invoked, in the order they are registered, if the request succeeds. They receive the returned data, a string containing the success code, and the `jqXHR` object.
- `complete` callbacks fire, in the order they are registered, when the request finishes, whether in failure or success. They receive the `jqXHR` object, as well as a string containing the success or error code.

For example, to make use of the returned HTML, we can implement a `success` handler:

```
$.ajax({
 url: 'ajax/test.html',
 success: function(data) {
  $('.result').html(data);
  alert('Load was performed.');
 }
});
```

Data Types

The `$.ajax()` function relies on the server to provide information about the retrieved data. If the server reports the return data as XML, the result can be traversed using normal XML methods or jQuery's selectors. If another type is detected, such as HTML in the example above, the data is treated as text.

Different data handling can be achieved by using the `dataType` option. Besides plain `xml`, the `dataType` can be `html`, `json`, `jsonp`, `script`, or `text`.

The `text` and `xml` types return the data with no processing. The data is simply passed on to the success handler, either through the `responseText` or `responseXML` property of the `jqXHR` object, respectively.

Note: We must ensure that the MIME type reported by the web server matches our choice of `dataType`. In particular, XML must be declared by the server as `text/xml` or `application/xml` for consistent results.

If `html` is specified, any embedded JavaScript inside the retrieved data is executed before the HTML is returned as a string. Similarly, `script` will execute the JavaScript that is pulled back from the server, then return nothing.

The `json` type parses the fetched data file as a JavaScript object and returns the constructed object as the result data. To do so, it uses `jQuery.parseJSON()` when the browser supports it; otherwise it uses a `Function`**constructor**. Malformed JSON data will throw a parse error (see json.org for more information). JSON data is convenient for communicating structured data in a way that is concise and easy for JavaScript to parse. If the fetched data file exists on a remote server, specify the `jsonp` type instead.

The `jsonp` type appends a query string parameter of `callback=?` to the URL. The server should prepend the JSON data with the callback name to form a valid JSONP response. We can specify a parameter name other than `callback` with the `jsonp` option to `$.ajax()`.

Note: JSONP is an extension of the JSON format, requiring some server-side code to detect and handle the query string parameter. More information about it can be found in the original post detailing its use.

When data is retrieved from remote servers (which is only possible using the `script` or `jsonp` data types), the `error` callbacks and global events will never be fired.

Sending Data to the Server

By default, Ajax requests are sent using the GET HTTP method. If the POST method is required, the method can be specified by setting a value for the `type` option. This option affects how the contents of the `data` option are sent to the server. POST data will always be transmitted to the server using UTF-8 charset, per the W3C XMLHTTPRequest standard.

The `data` option can contain either a query string of the form `key1=value1&key2=value2`, or a map of the form `{key1: 'value1', key2: 'value2'}`. If the latter form is used, the data is converted into a query string using `jQuery.param()` before it is sent. This processing can be circumvented by setting `processData` to `false`. The processing might be undesirable if you wish to send an XML object to the server; in this case, change the `contentType` option from `application/x-www-form-urlencoded` to a more appropriate MIME type.

Advanced Options

The `global` option prevents handlers registered using `.ajaxSend()`, `.ajaxError()`, and similar methods from firing when this request would trigger them. This can be useful to, for example, suppress a loading indicator that was implemented with `.ajaxSend()` if the requests are frequent and brief. With cross-domain script and JSONP requests, the global option is automatically set to `false`. See the descriptions of these methods below for more details. See the descriptions of these methods below for more details.

If the server performs HTTP authentication before providing a response, the user name and password pair can be sent via the `username` and `password` options.

Ajax requests are time-limited, so errors can be caught and handled to provide a better user experience. Request timeouts are usually either left at their default or set as a global default using `$.ajaxSetup()` rather than being overridden for specific requests with the `timeout` option.

By default, requests are always issued, but the browser may serve results out of its cache. To disallow use of the cached results, set `cache` to `false`. To cause the request to report failure if the asset has not been modified since the last request, set `ifModified` to `true`.

The `scriptCharset` allows the character set to be explicitly specified for requests that use a `<script>` tag (that is, a type of `script` or `jsonp`). This is useful if the script and host page have differing character sets.

The first letter in Ajax stands for "asynchronous," meaning that the operation occurs in parallel and the order of completion is not guaranteed. The `async` option to `$.ajax()` defaults to `true`, indicating that code execution can continue after the request is made. Setting this option to `false` (and thus making the call no longer asynchronous) is strongly discouraged, as it can cause the browser to become unresponsive.

The `$.ajax()` function returns the `XMLHttpRequest` object that it creates. Normally jQuery handles the creation of this object internally, but a custom function for manufacturing one can be specified using the `xhr` option. The returned object can generally be discarded, but does provide a lower-level interface for observing and manipulating the request. In particular, calling `.abort()` on the object will halt the request before it completes.

At present, due to a bug in Firefox where `.getAllResponseHeaders()` returns the empty string although `.getResponseHeader('Content-Type')` returns a non-empty string, automatically decoding JSON CORS responses in Firefox with jQuery is not supported.

A workaround to this is possible by overriding `jQuery.ajaxSettings.xhr` as follows:

```
var _super = jQuery.ajaxSettings.xhr;
jQuery.ajaxSettings.xhr = function () {
  var xhr = _super(),
    getAllResponseHeaders = xhr.getAllResponseHeaders;

  xhr.getAllResponseHeaders = function () {
    if ( getAllResponseHeaders() ) {
      return getAllResponseHeaders();
    }
    var allHeaders = "";
    $( ["Cache-Control", "Content-Language", "Content-Type",
        "Expires", "Last-Modified", "Pragma"] ).each(function (i, header_name) {

      if ( xhr.getResponseHeader( header_name ) ) {
        allHeaders += header_name + ": " + xhr.getResponseHeader( header_name ) + "n";
      }
      return allHeaders;
    });
  };
  return xhr;
};
```

Extending Ajax

As of jQuery 1.5, jQuery's Ajax implementation includes prefilters, converters, and transports that allow you to extend Ajax with a great deal of flexibility. For more information about these advanced features, see the Extending Ajax page.

Example

Save some data to the server and notify the user once it's complete.

```
$.ajax({
 type: "POST",
 url: "some.php",
 data: { name: "John", location: "Boston" }
}).done(function( msg ) {
 alert( "Data Saved: " + msg );
});
```

Example
Retrieve the latest version of an HTML page.

```
$.ajax({
 url: "test.html",
 cache: false
}).done(function( html ) {
 $("#results").append(html);
});
```

Example
Send an xml document as data to the server. By setting the processData option to `false`, the automatic conversion of data to strings is prevented.

```
var xmlDocument = [create xml document];
var xmlRequest = $.ajax({
 url: "page.php",
 processData: false,
 data: xmlDocument
});

xmlRequest.done(handleResponse);
```

Example
Send an id as data to the server, save some data to the server, and notify the user once it's complete. If the request fails, alert the user.

```
var menuId = $("ul.nav").first().attr("id");
var request = $.ajax({
 url: "script.php",
 type: "POST",
 data: {id : menuId},
 dataType: "html"
});

request.done(function(msg) {
 $("#log").html( msg );
});

request.fail(function(jqXHR, textStatus) {
 alert( "Request failed: " + textStatus );
});
```

Example
Load and execute a JavaScript file.

```
$.ajax({
 type: "GET",
 url: "test.js",
 dataType: "script"
});
```

Shorthand Methods

jQuery.post(url, [data], [success(data, textStatus, jqXHR)], [dataType])
Load data from the server using a HTTP POST request.

Arguments
url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.
dataType - The type of data expected from the server. Default: Intelligent Guess (xml, json, script, text, html).

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 type: 'POST',
 url: url,
 data: data,
 success: success,
 dataType: dataType
});
```

The `success` callback function is passed the returned data, which will be an XML root element or a text string depending on the MIME type of the response. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function is also passed a "jqXHR" object (in **jQuery 1.4**, it was passed the `XMLHttpRequest` object).

Most implementations will specify a success handler:

```
$.post('ajax/test.html', function(data) {
 $('.result').html(data);
});
```

This example fetches the requested HTML snippet and inserts it on the page.

Pages fetched with `POST` are never cached, so the `cache` and `ifModified` options in `jQuery.ajaxSetup()` have no effect on these requests.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.post()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.post()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
  // and remember the jqxhr object for this request
  var jqxhr = $.post("example.php", function() {
   alert("success");
  })
  .success(function() { alert("second success"); })
  .error(function() { alert("error"); })
  .complete(function() { alert("complete"); });

  // perform other work here ...

  // Set another completion function for the request above
  jqxhr.complete(function(){ alert("second complete"); });
```

Example

Request the test.php page, but ignore the return results.

```
$.post("test.php");
```

Example

Request the test.php page and send some additional data along (while still ignoring the return results).

```
$.post("test.php", { name: "John", time: "2pm" } );
```

Example
pass arrays of data to the server (while still ignoring the return results).

```
$.post("test.php", { 'choices[]': ["Jon", "Susan"] });
```

Example
send form data using ajax requests

```
$.post("test.php", $("#testform").serialize());
```

Example
Alert out the results from requesting test.php (HTML or XML, depending on what was returned).

```
$.post("test.php", function(data) {
  alert("Data Loaded: " + data);
 });
```

Example
Alert out the results from requesting test.php with an additional payload of data (HTML or XML, depending on what was returned).

```
$.post("test.php", { name: "John", time: "2pm" },
  function(data) {
   alert("Data Loaded: " + data);
  });
```

Example
Gets the test.php page content, store it in a XMLHttpResponse object and applies the process() JavaScript function.

```
$.post("test.php", { name: "John", time: "2pm" },
 function(data) {
  process(data);
 },
 "xml"
);
```

Example
Posts to the test.php page and gets contents which has been returned in json format (<?php echo json_encode(array("name"=>"John","time"=>"2pm")); ?>).

```
$.post("test.php", { "func": "getNameAndTime" },
 function(data){
  console.log(data.name); // John
  console.log(data.time); // 2pm
 }, "json");
```

Example
Post a form using ajax and put results in a div

```
/* attach a submit handler to the form */
 $("#searchForm").submit(function(event) {

  /* stop form from submitting normally */
  event.preventDefault();

  /* get some values from elements on the page: */
  var $form = $( this ),
    term = $form.find( 'input[name="s"]' ).val(),
    url = $form.attr( 'action' );

  /* Send the data using post and put the results in a div */
  $.post( url, { s: term },
   function( data ) {
     var content = $( data ).find( '#content' );
```

```
    $( "#result" ).empty().append( content );
  }
 );
});
```

jQuery.getScript(url, [success(script, textStatus, jqXHR)])

Load a JavaScript file from the server using a GET HTTP request, then execute it.

Arguments

url - A string containing the URL to which the request is sent.
success(script, textStatus, jqXHR) - A callback function that is executed if the request succeeds.

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
 dataType: "script",
 success: success
});
```

The script is executed in the global context, so it can refer to other variables and use jQuery functions. Included scripts can have some impact on the current page.

Success Callback

The callback is passed the returned JavaScript file. This is generally not useful as the script will already have run at this point.

```
$(".result").html("<p>Lorem ipsum dolor sit amet.</p>");
```

Scripts are included and run by referencing the file name:

```
$.getScript("ajax/test.js", function(data, textStatus, jqxhr) {
  console.log(data); //data returned
  console.log(textStatus); //success
  console.log(jqxhr.status); //200
  console.log('Load was performed.');
});
```

Handling Errors

As of jQuery 1.5, you may use .fail() to account for errors:

```
$.getScript("ajax/test.js")
.done(function(script, textStatus) {
 console.log( textStatus );
})
.fail(function(jqxhr, settings, exception) {
 $( "div.log" ).text( "Triggered ajaxError handler." );
});
```

Prior to jQuery 1.5, the global `.ajaxError()` callback event had to be used in order to handle `$.getScript()` errors:

```
$( "div.log" ).ajaxError(function(e, jqxhr, settings, exception) {
 if (settings.dataType=='script') {
  $(this).text( "Triggered ajaxError handler." );
 }
```

```
});
```

Caching Responses

Be default, `$.getScript()` sets the cache setting to `false`. This appends a timestamped query parameter to the request URL to ensure that the browser downloads the script each time it is requested. You can override this feature by setting the cache property globally using $.ajaxSetup():

```
$.ajaxSetup({
 cache: true
});
```

Alternatively, you could define a new method that uses the more flexible `$.ajax()` method.

Example

Define a $.cachedScript() method that allows fetching a cached script:

```
jQuery.cachedScript = function(url, options) {

 // allow user to set any option except for dataType, cache, and url
 options = $.extend(options || {}, {
  dataType: "script",
  cache: true,
  url: url
 });

 // Use $.ajax() since it is more flexible than $.getScript
 // Return the jqXHR object so we can chain callbacks
 return jQuery.ajax(options);
};

// Usage
$.cachedScript("ajax/test.js").done(function(script, textStatus) {
 console.log( textStatus );
});
```

Example

Load the official jQuery Color Animation plugin dynamically and bind some color animations to occur once the new functionality is loaded.

```
$.getScript("/scripts/jquery.color.js", function() {
 $("#go").click(function(){
  $(".block").animate( { backgroundColor: "pink" }, 1000)
   .delay(500)
   .animate( { backgroundColor: "blue" }, 1000);
 });
});
```

jQuery.getJSON(url, [data], [success(data, textStatus, jqXHR)])

Load JSON-encoded data from the server using a GET HTTP request.

Arguments

url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
```

```
 dataType: 'json',
 data: data,
 success: callback
});
```

Data that is sent to the server is appended to the URL as a query string. If the value of the `data` parameter is an object (map), it is converted to a string and url-encoded before it is appended to the URL.

Most implementations will specify a success handler:

```
$.getJSON('ajax/test.json', function(data) {
 var items = [];

 $.each(data, function(key, val) {
  items.push('<li id="' + key + '">' + val + '</li>');
 });

 $('<ul/>', {
  'class': 'my-new-list',
  html: items.join('')
 }).appendTo('body');
});
```

This example, of course, relies on the structure of the JSON file:

```
{
 "one": "Singular sensation",
 "two": "Beady little eyes",
 "three": "Little birds pitch by my doorstep"
}
```

Using this structure, the example loops through the requested data, builds an unordered list, and appends it to the body.

The `success` callback is passed the returned data, which is typically a JavaScript object or array as defined by the JSON structure and parsed using the `$.parseJSON()` method. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function receives a "jqXHR" object (in **jQuery 1.4**, it received the `XMLHttpRequest` object). However, since JSONP and cross-domain GET requests do not use XHR, in those cases the `jqXHR` and `textStatus` parameters passed to the success callback are undefined.

Important: As of jQuery 1.4, if the JSON file contains a syntax error, the request will usually fail silently. Avoid frequent hand-editing of JSON data for this reason. JSON is a data-interchange format with syntax rules that are stricter than those of JavaScript's object literal notation. For example, all strings represented in JSON, whether they are properties or values, must be enclosed in double-quotes. For details on the JSON format, see http://json.org/.

JSONP

If the URL includes the string "callback=?" (or similar, as defined by the server-side API), the request is treated as JSONP instead. See the discussion of the `jsonp` data type in `$.ajax()` for more details.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.getJSON()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.getJSON()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.getJSON("example.json", function() {
 alert("success");
})
.success(function() { alert("second success"); })
.error(function() { alert("error"); })
.complete(function() { alert("complete"); });

// perform other work here ...

// Set another completion function for the request above
jqxhr.complete(function(){ alert("second complete"); });
```

Example
Loads the four most recent cat pictures from the Flickr JSONP API.

```
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?",
 {
  tags: "cat",
  tagmode: "any",
  format: "json"
 },
 function(data) {
  $.each(data.items, function(i,item){
   $("<img/>").attr("src", item.media.m).appendTo("#images");
   if ( i == 3 ) return false;
  });
 });
```

Example
Load the JSON data from test.js and access a name from the returned JSON data.

```
$.getJSON("test.js", function(json) {
  alert("JSON Data: " + json.users[3].name);
 });
```

Example
Load the JSON data from test.js, passing along additional data, and access a name from the returned JSON data.

```
$.getJSON("test.js", { name: "John", time: "2pm" }, function(json) {
  alert("JSON Data: " + json.users[3].name);
  });
```

jQuery.get(url, [data], [success(data, textStatus, jqXHR)], [dataType])
Load data from the server using a HTTP GET request.

Arguments
url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.
dataType - The type of data expected from the server. Default: Intelligent Guess (xml, json, script, or html).

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
 data: data,
```

```
 success: success,
 dataType: dataType
});
```

The `success` callback function is passed the returned data, which will be an XML root element, text string, JavaScript file, or JSON object, depending on the MIME type of the response. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function is also passed a "jqXHR" object (in **jQuery 1.4**, it was passed the `XMLHttpRequest` object). However, since JSONP and cross-domain GET requests do not use XHR, in those cases the `(j)XHR` and `textStatus` parameters passed to the success callback are undefined.

Most implementations will specify a success handler:

```
$.get('ajax/test.html', function(data) {
 $('.result').html(data);
 alert('Load was performed.');
});
```

This example fetches the requested HTML snippet and inserts it on the page.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.get()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.get()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
 // and remember the jqxhr object for this request
 var jqxhr = $.get("example.php", function() {
  alert("success");
 })
 .success(function() { alert("second success"); })
 .error(function() { alert("error"); })
 .complete(function() { alert("complete"); });

 // perform other work here ...

 // Set another completion function for the request above
 jqxhr.complete(function(){ alert("second complete"); });
```

Example

Request the test.php page, but ignore the return results.

```
$.get("test.php");
```

Example

Request the test.php page and send some additional data along (while still ignoring the return results).

```
$.get("test.php", { name: "John", time: "2pm" } );
```

Example

pass arrays of data to the server (while still ignoring the return results).

```
$.get("test.php", { 'choices[]': ["Jon", "Susan"]} );
```

Example
Alert out the results from requesting test.php (HTML or XML, depending on what was returned).

```
$.get("test.php", function(data){
alert("Data Loaded: " + data);
});
```

Example
Alert out the results from requesting test.cgi with an additional payload of data (HTML or XML, depending on what was returned).

```
$.get("test.cgi", { name: "John", time: "2pm" },
  function(data){
   alert("Data Loaded: " + data);
  });
```

Example
Gets the test.php page contents, which has been returned in json format (<?php echo json_encode(array("name"=>"John","time"=>"2pm")); ?>), and adds it to the page.

```
$.get("test.php",
  function(data){
   $('body').append( "Name: " + data.name ) // John
        .append( "Time: " + data.time ); // 2pm
  }, "json");
```

load(url, data, [complete(responseText, textStatus, XMLHttpRequest)])
Load data from the server and place the returned HTML into the matched element.

Arguments
url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
complete(responseText, textStatus, XMLHttpRequest) - A callback function that is executed when the request completes.

Note: The event handling suite also has a method named `.load()`. jQuery determines which method to fire based on the set of arguments passed to it.

This method is the simplest way to fetch data from the server. It is roughly equivalent to `$.get(url, data, success)` except that it is a method rather than global function and it has an implicit callback function. When a successful response is detected (i.e. when `textStatus` is "success" or "notmodified"), `.load()` sets the HTML contents of the matched element to the returned data. This means that most uses of the method can be quite simple:

```
$('#result').load('ajax/test.html');
```

Callback Function

If a "complete" callback is provided, it is executed after post-processing and HTML insertion has been performed. The callback is fired once for each element in the jQuery collection, and `this` is set to each DOM element in turn.

```
$('#result').load('ajax/test.html', function() {
 alert('Load was performed.');
});
```

In the two examples above, if the current document does not contain an element with an ID of "result," the `.load()` method is not executed.

Request Method

The POST method is used if data is provided as an object; otherwise, GET is assumed.

Loading Page Fragments

The .load() method, unlike $.get(), allows us to specify a portion of the remote document to be inserted. This is achieved with a special syntax for the url parameter. If one or more space characters are included in the string, the portion of the string following the first space is assumed to be a jQuery selector that determines the content to be loaded.

We could modify the example above to use only part of the document that is fetched:

```
$('#result').load('ajax/test.html #container');
```

When this method executes, it retrieves the content of ajax/test.html, but then jQuery parses the returned document to find the element with an ID of container. This element, along with its contents, is inserted into the element with an ID of result, and the rest of the retrieved document is discarded.

jQuery uses the browser's .innerHTML property to parse the retrieved document and insert it into the current document. During this process, browsers often filter elements from the document such as <html>, <title>, or <head> elements. As a result, the elements retrieved by .load() may not be exactly the same as if the document were retrieved directly by the browser.

Script Execution

When calling .load() using a URL without a suffixed selector expression, the content is passed to .html() prior to scripts being removed. This executes the script blocks before they are discarded. If .load() is called with a selector expression appended to the URL, however, the scripts are stripped out prior to the DOM being updated, and thus are *not* executed. An example of both cases can be seen below:

Here, any JavaScript loaded into #a as a part of the document will successfully execute.

```
$('#a').load('article.html');
```

However, in the following case, script blocks in the document being loaded into #b are stripped out and not executed:

```
$('#b').load('article.html #target');
```

Example

Load the main page's footer navigation into an ordered list.

```
$("#new-nav").load("/ #jq-footerNavigation li");
```

Example

Display a notice if the Ajax request encounters an error.

```
$("#success").load("/not-here.php", function(response, status, xhr) {
 if (status == "error") {
  var msg = "Sorry but there was an error: ";
  $("#error").html(msg + xhr.status + " " + xhr.statusText);
 }
});
```

Example

Load the feeds.html file into the div with the ID of feeds.

```
$("#feeds").load("feeds.html");
```

Example

pass arrays of data to the server.

```
$("#objectID").load("test.php", { 'choices[]': ["Jon", "Susan"] } );
```

Example

Same as above, but will POST the additional parameters to the server and a callback that is executed when the server is finished responding.

```
$("#feeds").load("feeds.php", {limit: 25}, function(){
alert("The last 25 entries in the feed have been loaded");
});
```

Attributes

removeProp(propertyName)

Remove a property for the set of matched elements.

Arguments

propertyName - The name of the property to set.

The `.removeProp()` method removes properties set by the `.prop()` method.

With some built-in properties of a DOM element or `window` object, browsers may generate an error if an attempt is made to remove the property. jQuery first assigns the value `undefined` to the property and ignores any error the browser generates. In general, it is only necessary to remove custom properties that have been set on an object, and not built-in (native) properties.

Note: Do not use this method to remove native properties such as checked, disabled, or selected. This will remove the property completely and, once removed, cannot be added again to element. Use `.prop()` to set these properties to `false` instead.

Example

Set a numeric property on a paragraph and then remove it.

```
var $para = $("p");
$para.prop("luggageCode", 1234);
$para.append("The secret luggage code is: ", String($para.prop("luggageCode")), ". ");
$para.removeProp("luggageCode");
$para.append("Now the secret luggage code is: ", String($para.prop("luggageCode")), ". ");
```

prop(propertyName)

Get the value of a property for the first element in the set of matched elements.

Arguments

propertyName - The name of the property to get.

The `.prop()` method gets the property value for only the *first* element in the matched set. It returns `undefined` for the value of a property that has not been set, or if the matched set has no elements. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

The difference between *attributes* and *properties* can be important in specific situations. **Before jQuery 1.6**, the `.attr()` method sometimes took property values into account when retrieving some attributes, which could cause inconsistent behavior. **As of jQuery 1.6**, the `.prop()` method provides a way to explicitly retrieve property values, while `.attr()` retrieves attributes.

For example, `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, and `defaultSelected` should be retrieved and set with the `.prop()` method. Prior to jQuery 1.6, these properties were retrievable with the `.attr()` method, but this was not within the scope of `attr`. These do not have corresponding attributes and are only properties.

Concerning boolean attributes, consider a DOM element defined by the HTML markup `<input type="checkbox" checked="checked" />`, and assume it is in a JavaScript variable named `elem`:

`elem.checked`	`true`	(Boolean) Will change with checkbox state	
`$(elem).prop("checked")`	`true`	(Boolean) Will change with checkbox state	
`elem.getAttribute("checked")`	`"checked"`	(String) Initial state of the checkbox; does not change	
`$(elem).attr("checked")`	*(1.6)*	`"checked"`	(String) Initial s
`$(elem).attr("checked")`	*(1.6.1+)*	`"checked"`	(String) Will ch
`$(elem).attr("checked")`	*(pre-1.6)*	`true`	(Boolean) Cha

According to the W3C forms specification, the `checked` attribute is a *boolean attribute*, which means the corresponding property is true if the attribute is present at all-even if, for example, the attribute has no value or an empty string value. The preferred cross-browser-compatible way to determine if a

checkbox is checked is to check for a "truthy" value on the element's property using one of the following:

- `if ( elem.checked )`
- `if ( $(elem).prop("checked") )`
- `if ( $(elem).is(":checked") )`

If using jQuery 1.6, the code `if ( $(elem).attr("checked") )` will retrieve the actual content *attribute*, which does not change as the checkbox is checked and unchecked. It is meant only to store the default or initial value of the checked property. To maintain backwards compatability, the `.attr()` method in jQuery 1.6.1+ will retrieve and update the property for you so no code for boolean attributes is required to be changed to `.prop()`. Nevertheless, the preferred way to retrieve a checked value is with one of the options listed above. To see how this works in the latest jQuery, check/uncheck the checkbox in the example below.

Example

Display the checked property and attribute of a checkbox as it changes.

```
$("input").change(function() {
 var $input = $(this);
 $("p").html(".attr('checked'): <b>" + $input.attr('checked') + "</b><br>"
       + ".prop('checked'): <b>" + $input.prop('checked') + "</b><br>"
       + ".is(':checked'): <b>" + $input.is(':checked') ) + "</b>";
}).change();
```

prop(propertyName, value)

Set one or more properties for the set of matched elements.

Arguments

propertyName - The name of the property to set.
value - A value to set for the property.

The `.prop()` method is a convenient way to set the value of properties-especially when setting multiple properties, using values returned by a function, or setting values on multiple elements at once. It should be used when setting `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, or `defaultSelected`. Since jQuery 1.6, these properties can no longer be set with the `.attr()` method. They do not have corresponding attributes and are only properties.

Properties generally affect the dynamic state of a DOM element without changing the serialized HTML attribute. Examples include the `value` property of input elements, the `disabled` property of inputs and buttons, or the `checked` property of a checkbox. The `.prop()` method should be used to set disabled and checked instead of the `.attr()` method. The `.val()` method should be used for getting and setting value.

```
$("input").prop("disabled", false);
$("input").prop("checked", true);
$("input").val("someValue");
```

Important: the `.removeProp()` method should not be used to set these properties to false. Once a native property is removed, it cannot be added again. See `.removeProp()` for more information.

Computed property values

By using a function to set properties, you can compute the value based on other properties of the element. For example, to toggle all checkboxes based off their individual values:

```
$("input[type='checkbox']").prop("checked", function( i, val ) {
 return !val;
});
```

Note: If nothing is returned in the setter function (ie. `function(index, prop){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example

Disable all checkboxes on the page.

```
$("input[type='checkbox']").prop({
 disabled: true
});
```

val()

Get the current value of the first element in the set of matched elements.

The `.val()` method is primarily used to get the values of form elements such as `input`, `select` and `textarea`. In the case of `<select multiple="multiple">` elements, the `.val()` method returns an array containing each selected option; if no option is selected, it returns `null`.

For selects and checkboxes, you can also use the :selected and :checked selectors to get at values, for example:

```
$('select.foo option:selected').val();  // get the value from a dropdown select
$('select.foo').val();          // get the value from a dropdown select even easier
$('input:checkbox:checked').val();    // get the value from a checked checkbox
$('input:radio[name=bar]:checked').val(); // get the value from a set of radio buttons
```

Note: At present, using `.val()` on textarea elements strips carriage return characters from the browser-reported value. When this value is sent to the server via XHR however, carriage returns are preserved (or added by browsers which do not include them in the raw value). A workaround for this issue can be achieved using a valHook as follows:

```
$.valHooks.textarea = {
    get: function( elem ) {
        return elem.value.replace( /r?n/g, "rn" );
    }
};
```

Example

Get the single value from a single select and an array of values from a multiple select and display their values.

```
function displayVals() {
   var singleValues = $("#single").val();
   var multipleValues = $("#multiple").val() || [];
   $("p").html("<b>Single:</b> " +
         singleValues +
         " <b>Multiple:</b> " +
         multipleValues.join(", "));
  }

  $("select").change(displayVals);
  displayVals();
```

Example

Find the value of an input box.

```
$("input").keyup(function () {
   var value = $(this).val();
   $("p").text(value);
  }).keyup();
```

val(value)

Set the value of each element in the set of matched elements.

Arguments

value - A string of text or an array of strings corresponding to the value of each matched element to set as selected/checked.

This method is typically used to set the values of form fields.

Passing an array of element values allows matching <input type="checkbox">, <input type="radio"> and <option>s inside of n <select multiple="multiple"> to be selected. In the case of <input type="radio">s that are part of a radio group and <select multiple="multiple"> the other elements will be deselected.

The .val() method allows us to set the value by passing in a function. As of jQuery 1.4, the function is passed two arguments, the current element's index and its current value:

```
$('input:text.items').val(function( index, value ) {
 return value + ' ' + this.className;
});
```

This example appends the string " items" to the text inputs' values.

Example
Set the value of an input box.

```
$("button").click(function () {
   var text = $(this).text();
   $("input").val(text);
  });
```

Example
Use the function argument to modify the value of an input box.

```
$('input').bind('blur', function() {
  $(this).val(function( i, val ) {
   return val.toUpperCase();
  });
 });
```

Example
Set a single select, a multiple select, checkboxes and a radio button .

```
$("#single").val("Single2");
  $("#multiple").val(["Multiple2", "Multiple3"]);
  $("input").val(["check1","check2", "radio1" ]);
```

html()
Get the HTML contents of the first element in the set of matched elements.

This method is not available on XML documents.

In an HTML document, .html() can be used to get the contents of any element. If the selector expression matches more than one element, only the first match will have its HTML content returned. Consider this code:

```
$('div.demo-container').html();
```

In order for the following 's content to be retrieved, it would have to be the first one with class="demo-container" in the document:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
</div>
```

The result would look like this:

```
<div class="demo-box">Demonstration Box</div>
```

This method uses the browser's innerHTML property. Some browsers may not return HTML that exactly replicates the HTML source in an original document. For example, Internet Explorer sometimes leaves off the quotes around attribute values if they contain only alphanumeric characters.

Example
Click a paragraph to convert it from html to text.

```
$("p").click(function () {
   var htmlStr = $(this).html();
   $(this).text(htmlStr);
  });
```

html(htmlString)
Set the HTML contents of each element in the set of matched elements.

Arguments
htmlString - A string of HTML to set as the content of each matched element.

The `.html()` method is not available in XML documents.

When `.html()` is used to set an element's content, any content that was in that element is completely replaced by the new content. Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
</div>
```

The content of `<div class="demo-container">` can be set like this:

```
$('div.demo-container')
 .html('<p>All new content. <em>You bet!</em></p>');
```

That line of code will replace everything inside `<div class="demo-container">`:

```
<div class="demo-container">
 <p>All new content. <em>You bet!</em></p>
</div>
```

As of jQuery 1.4, the `.html()` method allows the HTML content to be set by passing in a function.

```
$('div.demo-container').html(function() {
 var emph = '<em>' + $('p').length + ' paragraphs!</em>';
 return '<p>All new content for ' + emph + '</p>';
});
```

Given a document with six paragraphs, this example will set the HTML of `<div class="demo-container">` to `<p>All new content for <em>6 paragraphs!</em></p>`.

This method uses the browser's `innerHTML` property. Some browsers may not generate a DOM that exactly replicates the HTML source provided. For example, Internet Explorer prior to version 8 will convert all `href` properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate compatibility layer.

Note: In Internet Explorer up to and including version 9, setting the text content of an HTML element may corrupt the text nodes of its children that are being removed from the document as a result of the operation. If you are keeping references to these DOM elements and need them to be unchanged, use `.empty().html(string)` instead of `.html(string)` so that the elements are removed from the document before the new string is assigned to the element.

Example
Add some html to each div.

```
$("div").html("<span class='red'>Hello <b>Again</b></span>");
```

Example
Add some html to each div then immediately do further manipulations to the inserted html.

```
$("div").html("<b>Wow!</b> Such excitement...");
```

```
  $("div b").append(document.createTextNode("!!!"))
       .css("color", "red");
```

toggleClass(className)

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Arguments

className - One or more class names (separated by spaces) to be toggled for each element in the matched set.

This method takes one or more class names as its parameter. In the first version, if an element in the matched set of elements already has the class, then it is removed; if an element does not have the class, then it is added. For example, we can apply `.toggleClass()` to a simple `<div>`:

```
<div class="tumble">Some text.</div>
```

The first time we apply `$('div.tumble').toggleClass('bounce')`, we get the following:

```
<div class="tumble bounce">Some text.</div>
```

The second time we apply `$('div.tumble').toggleClass('bounce')`, the `<div>` class is returned to the single `tumble` value:

```
<div class="tumble">Some text.</div>
```

Applying `.toggleClass('bounce spin')` to the same `<div>` alternates between `<div class="tumble bounce spin">` and `<div class="tumble">`.

The second version of `.toggleClass()` uses the second parameter for determining whether the class should be added or removed. If this parameter's value is `true`, then the class is added; if `false`, the class is removed. In essence, the statement:

```
$('#foo').toggleClass(className, addOrRemove);
```

is equivalent to:

```
if (addOrRemove) {
  $('#foo').addClass(className);
 }
 else {
  $('#foo').removeClass(className);
 }
```

As of jQuery 1.4, if no arguments are passed to `.toggleClass()`, all class names on the element the first time `.toggleClass()` is called will be toggled. Also as of jQuery 1.4, the class name to be toggled can be determined by passing in a function.

```
$('div.foo').toggleClass(function() {
 if ($(this).parent().is('.bar')) {
  return 'happy';
 } else {
  return 'sad';
 }
});
```

This example will toggle the `happy` class for `<div class="foo">` elements if their parent element has a class of `bar`; otherwise, it will toggle the `sad` class.

Example

Toggle the class 'highlight' when a paragraph is clicked.

```
$("p").click(function () {
   $(this).toggleClass("highlight");
```

```
  });
```

Example
Add the "highlight" class to the clicked paragraph on every third click of that paragraph, remove it every first and second click.

```
var count = 0;
$("p").each(function() {
 var $thisParagraph = $(this);
 var count = 0;
 $thisParagraph.click(function() {
  count++;
  $thisParagraph.find("span").text('clicks: ' + count);
  $thisParagraph.toggleClass("highlight", count % 3 == 0);
 });
});
```

Example
Toggle the class name(s) indicated on the buttons for each div.

```
var cls = ['', 'a', 'a b', 'a b c'];
var divs = $('div.wrap').children();
var appendClass = function() {
 divs.append(function() {
  return '<div>' + (this.className || 'none') + '</div>';
 });
};

appendClass();

$('button').bind('click', function() {
 var tc = this.className || undefined;
 divs.toggleClass(tc);
 appendClass();
});

$('a').bind('click', function(event) {
 event.preventDefault();
 divs.empty().each(function(i) {
  this.className = cls[i];
 });
 appendClass();
});
```

removeClass([className])
Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Arguments
className - One or more space-separated classes to be removed from the class attribute of each matched element.

If a class name is included as a parameter, then only that class will be removed from the set of matched elements. If no class names are specified in the parameter, all classes will be removed.

More than one class may be removed at a time, separated by a space, from the set of matched elements, like so:

```
$('p').removeClass('myClass yourClass')
```

This method is often used with `.addClass()` to switch elements' classes from one to another, like so:

```
$('p').removeClass('myClass noClass').addClass('yourClass');
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

To replace all existing classes with another class, we can use `.attr('class', 'newClass')` instead.

As of jQuery 1.4, the `.removeClass()` method allows us to indicate the class to be removed by passing in a function.

```
$('li:last').removeClass(function() {
     return $(this).prev().attr('class');
    });
```

This example removes the class name of the penultimate `<li>` from the last `<li>`.

Example
Remove the class 'blue' from the matched elements.

```
$("p:even").removeClass("blue");
```

Example
Remove the class 'blue' and 'under' from the matched elements.

```
$("p:odd").removeClass("blue under");
```

Example
Remove all the classes from the matched elements.

```
$("p:eq(1)").removeClass();
```

hasClass(className)
Determine whether any of the matched elements are assigned the given class.

Arguments
className - The class name to search for.

Elements may have more than one class assigned to them. In HTML, this is represented by separating the class names with a space:

```
<div id="mydiv" class="foo bar"></div>
```

The `.hasClass()` method will return `true` if the class is assigned to an element, even if other classes also are. For example, given the HTML above, the following will return `true`:

```
$('#mydiv').hasClass('foo')
```

As would:

```
$('#mydiv').hasClass('bar')
```

While this would return `false`:

```
$('#mydiv').hasClass('quux')
```

Example
Looks for the paragraph that contains 'selected' as a class.

```
$("div#result1").append($("p:first").hasClass("selected").toString());
$("div#result2").append($("p:last").hasClass("selected").toString());
$("div#result3").append($("p").hasClass("selected").toString());
```

removeAttr(attributeName)
Remove an attribute from each element in the set of matched elements.

Arguments

attributeName - An attribute to remove; as of version 1.7, it can be a space-separated list of attributes.

The `.removeAttr()` method uses the JavaScript `removeAttribute()` function, but it has the advantage of being able to be called directly on a jQuery object and it accounts for different attribute naming across browsers.

Note: Removing an inline `onclick` event handler using `.removeAttr()` doesn't achieve the desired effect in Internet Explorer 6, 7, or 8. To avoid potential problems, use `.prop()` instead:

```
$element.prop("onclick", null);
console.log("onclick property: ", $element[0].onclick);
```

Example
Clicking the button enables the input next to it.

```
(function() {
 var inputTitle = $("input").attr("title");
 $("button").click(function () {
  var input = $(this).next();

  if ( input.attr("title") == inputTitle ) {
   input.removeAttr("title")
  } else {
   input.attr("title", inputTitle);
  }

  $("#log").html( "input title is now " + input.attr("title") );
 });
})();
```

attr(attributeName)
Get the value of an attribute for the first element in the set of matched elements.

Arguments
attributeName - The name of the attribute to get.

The `.attr()` method gets the attribute value for only the *first* element in the matched set. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

As of jQuery 1.6, the `.attr()` method returns `undefined` for attributes that have not been set. In addition, `.attr()` should not be used on plain objects, arrays, the window, or the document. To retrieve and change DOM properties, use the .prop() method.

Using jQuery's `.attr()` method to get the value of an element's attribute has two main benefits:
- **Convenience**: It can be called directly on a jQuery object and chained to other jQuery methods.
- **Cross-browser consistency**: The values of some attributes are reported inconsistently across browsers, and even across versions of a single browser. The `.attr()` method reduces such inconsistencies.

Note: Attribute values are strings with the exception of a few attributes such as value and tabindex.

Example
Find the title attribute of the first in the page.

```
var title = $("em").attr("title");
 $("div").text(title);
```

attr(attributeName, value)
Set one or more attributes for the set of matched elements.

Arguments
attributeName - The name of the attribute to set.

value - A value to set for the attribute.

The `.attr()` method is a convenient way to set the value of attributes-especially when setting multiple attributes or using values returned by a function. Consider the following image:

```
<img id="greatphoto" src="brush-seller.jpg" alt="brush seller" />
```

Setting a simple attribute

To change the `alt` attribute, simply pass the name of the attribute and its new value to the `.attr()` method:

```
$('#greatphoto').attr('alt', 'Beijing Brush Seller');
```

Add an attribute the same way:

```
$('#greatphoto')
.attr('title', 'Photo by Kelly Clark');
```

Setting several attributes at once

To change the `alt` attribute and add the `title` attribute at the same time, pass both sets of names and values into the method at once using a map (JavaScript object literal). Each key-value pair in the map adds or modifies an attribute:

```
$('#greatphoto').attr({
 alt: 'Beijing Brush Seller',
 title: 'photo by Kelly Clark'
});
```

When setting multiple attributes, the quotes around attribute names are optional.

WARNING: When setting the 'class' attribute, you must always use quotes!

Note: jQuery prohibits changing the `type` attribute on an `<input>` or `<button>` element and will throw an error in all browsers. This is because the `type` attribute cannot be changed in Internet Explorer.

Computed attribute values

By using a function to set attributes, you can compute the value based on other properties of the element. For example, to concatenate a new value with an existing value:

```
$('#greatphoto').attr('title', function(i, val) {
 return val + ' - photo by Kelly Clark'
});
```

This use of a function to compute attribute values can be particularly useful when modifying the attributes of multiple elements at once.

Note: If nothing is returned in the setter function (ie. `function(index, attr){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example

Set some attributes for all <img>s in the page.

```
$("img").attr({
 src: "/images/hat.gif",
 title: "jQuery",
 alt: "jQuery Logo"
});
$("div").text($("img").attr("alt"));
```

Example

Set the id for divs based on the position in the page.

```
$("div").attr("id", function (arr) {
 return "div-id" + arr;
})
.each(function () {
 $("span", this).html("(ID = '<b>" + this.id + "</b>')");
});
```

Example
Set the src attribute from title attribute on the image.

```
$("img").attr("src", function() {
  return "/images/" + this.title;
});
```

addClass(className)
Adds the specified class(es) to each of the set of matched elements.

Arguments
className - One or more class names to be added to the class attribute of each matched element.

It's important to note that this method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.

More than one class may be added at a time, separated by a space, to the set of matched elements, like so:

```
$("p").addClass("myClass yourClass");
```

This method is often used with `.removeClass()` to switch elements' classes from one to another, like so:

```
$("p").removeClass("myClass noClass").addClass("yourClass");
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

As of jQuery 1.4, the `.addClass()` method's argument can receive a function.

```
$("ul li:last").addClass(function(index) {
 return "item-" + index;
});
```

Given an unordered list with five `<li>` elements, this example adds the class "item-4" to the last `<li>`.

Example
Adds the class "selected" to the matched elements.

```
$("p:last").addClass("selected");
```

Example
Adds the classes "selected" and "highlight" to the matched elements.

```
$("p:last").addClass("selected highlight");
```

Example
Pass in a function to `.addClass()` to add the "green" class to a div that already has a "red" class.

```
$("div").addClass(function(index, currentClass) {
  var addedClass;

  if ( currentClass === "red" ) {
   addedClass = "green";
   $("p").text("There is one green div");
```

```
  }

  return addedClass;
});
```

Callbacks Object

callbacks.fired()
Determine if the callbacks have already been called at least once.

Example

Using `callbacks.fired()` to determine if the callbacks in a list have been called at least once:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'
callbacks.fire( 'world '); // outputs: 'foo: world'

// test to establish if the callbacks have been called
console.log( callbacks.fired() );
```

jQuery.Callbacks(flags)
A multi-purpose callbacks list object that provides a powerful way to manage callback lists.

Arguments
flags - An optional list of space-separated flags that change how the callback list behaves.

The `$.Callbacks()` function is internally used to provide the base functionality behind the jQuery `$.ajax()` and `$.Deferred()` components. It can be used as a similar base to define functionality for new components.

`$.Callbacks()` support a number of methods including `callbacks.add()`,`callbacks.remove()`, `callbacks.fire()` and `callbacks.disable()`.

Getting started

The following are two sample methods named `fn1` and `fn2`:

```
function fn1( value ){
  console.log( value );
}

function fn2( value ){
  fn1("fn2 says:" + value);
  return false;
}
```

These can be added as callbacks to a `$.Callbacks` list and invoked follows:

```
var callbacks = $.Callbacks();
callbacks.add( fn1 );
callbacks.fire( "foo!" ); // outputs: foo!

callbacks.add( fn2 );
callbacks.fire( "bar!" ); // outputs: bar!, fn2 says: bar!
```

The result of this is that it becomes simple to construct complex lists of callbacks where input values can be passed through to as many functions as needed with ease.

Two specific methods were being used above: `.add()` and `.fire()` .add() supports adding new callbacks to the callback list, whilst fire() provides a way to pass arguments to be processed by the callbacks in the same list.

Another method supported by `$.Callbacks` is remove(), which has the ability to remove a particular callback from the callback list. Here"s a practical example of .remove() being used:

```
var callbacks = $.Callbacks();
callbacks.add( fn1 );
callbacks.fire( "foo!" ); // outputs: foo!

callbacks.add( fn2 );
callbacks.fire( "bar!" ); // outputs: bar!, fn2 says: bar!

callbacks.remove(fn2);
callbacks.fire( "foobar" );

// only outputs foobar, as fn2 has been removed.
```

Supported Flags

The `flags` argument is an optional argument to `$.Callbacks()`, structured as a list of space-separated strings that change how the callback list behaves (eg. `$.Callbacks( 'unique stopOnFalse' )`).

Possible flags:

- `once`: Ensures the callback list can only be fired once (like a Deferred).
- `memory`: Keep track of previous values and will call any callback added after the list has been fired right away with the latest "memorized" values (like a Deferred).
- `unique`: Ensures a callback can only be added once (so there are no duplicates in the list).
- `stopOnFalse`: Interrupts callings when a callback returns false.

By default a callback list will act like an event callback list and can be "fired" multiple times.

For examples of how `flags` should ideally be used, see below:

`$.Callbacks( 'once' )`:

```
var callbacks = $.Callbacks( "once" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
```

```
output:
foo
*/
```

$.Callbacks('memory'):

```
var callbacks = $.Callbacks( "memory" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
fn2 says:foo
bar
fn2 says:bar
foobar
*/
```

$.Callbacks('unique'):

```
var callbacks = $.Callbacks( "unique" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn1 ); // repeat addition
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
bar
fn2 says:bar
foobar
*/
```

$.Callbacks('stopOnFalse'):

```
function fn1( value ){
  console.log( value );
  return false;
}

function fn2( value ){
  fn1("fn2 says:" + value);
  return false;
}

var callbacks = $.Callbacks( "stopOnFalse");
callbacks.add( fn1 );
```

```
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
bar
foobar
*/
```

Because $.Callbacks() supports a list of flags rather than just one, setting several flags has a cumulative effect similar to "&&". This means it's possible to combine flags to create callback lists that are say, both *unique* and *ensure if list was already fired, adding more callbacks will have it called with the latest fired value* (i.e. `$.Callbacks("unique memory")`).

`$.Callbacks( 'unique memory' )`:

```
function fn1( value ){
  console.log( value );
  return false;
}

function fn2( value ){
  fn1("fn2 says:" + value);
  return false;
}

var callbacks = $.Callbacks( "unique memory" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn1 ); // repeat addition
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.add( fn2 );
callbacks.fire( "baz" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
fn2 says:foo
bar
fn2 says:bar
baz
fn2 says:baz
foobar
*/
```

Flag combinations are internally used with `$.Callbacks()` in jQuery for the `.done()` and `.fail()` buckets on a Deferred - both of which use `$.Callbacks('memory once')`.

$.Callbacks methods can also be detached, should there be a need to define short-hand versions for convenience:

```
var callbacks = $.Callbacks(),
  add = callbacks.add,
```

```
  remove = callbacks.remove,
  fire = callbacks.fire;

add( fn1 );
fire( "hello world");
remove( fn1 );
```

$.Callbacks, $.Deferred and Pub/Sub

The general idea behind pub/sub (the Observer pattern) is the promotion of loose coupling in applications. Rather than single objects calling on the methods of other objects, an object instead subscribes to a specific task or activity of another object and is notified when it occurs. Observers are also called Subscribers and we refer to the object being observed as the Publisher (or the subject). Publishers notify subscribers when events occur

As a demonstration of the component-creation capabilities of `$.Callbacks()`, it's possible to implement a Pub/Sub system using only callback lists. Using `$.Callbacks` as a topics queue, a system for publishing and subscribing to topics can be implemented as follows:

```
var topics = {};

jQuery.Topic = function( id ) {
  var callbacks,
    method,
    topic = id && topics[ id ];
  if ( !topic ) {
    callbacks = jQuery.Callbacks();
    topic = {
      publish: callbacks.fire,
      subscribe: callbacks.add,
      unsubscribe: callbacks.remove
    };
    if ( id ) {
      topics[ id ] = topic;
    }
  }
  return topic;
};
```

This can then be used by parts of your application to publish and subscribe to events of interest quite easily:

```
// Subscribers
$.Topic( "mailArrived" ).subscribe( fn1 );
$.Topic( "mailArrived" ).subscribe( fn2 );
$.Topic( "mailSent" ).subscribe( fn1 );

// Publisher
$.Topic( "mailArrived" ).publish( "hello world!" );
$.Topic( "mailSent" ).publish( "woo! mail!" );

// Here, "hello world!" gets pushed to fn1 and fn2
// when the "mailArrived" notification is published
// with "woo! mail!" also being pushed to fn1 when
// the "mailSent" notification is published.

/*
output:
hello world!
fn2 says: hello world!
woo! mail!
*/
```

Whilst this is useful, the implementation can be taken further. Using `$.Deferreds`, it's possible to ensure publishers only publish notifications for subscribers once particular tasks have been completed (resolved). See the below code sample for some further comments on how this could be used in practice:

```
// subscribe to the mailArrived notification
$.Topic( "mailArrived" ).subscribe( fn1 );

// create a new instance of Deferreds
var dfd = $.Deferred();

// define a new topic (without directly publishing)
var topic = $.Topic( "mailArrived" );

// when the deferred has been resolved, publish a
// notification to subscribers
dfd.done( topic.publish );

// Here the Deferred is being resolved with a message
// that will be passed back to subscribers. It's possible to
// easily integrate this into a more complex routine
// (eg. waiting on an ajax call to complete) so that
// messages are only published once the task has actually
// finished.
dfd.resolve( "its been published!" );
```

callbacks.locked()

Determine if the callbacks list has been locked.

Example

Using `callbacks.locked()` to determine the lock-state of a callback list:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value);
}

var callbacks = $.Callbacks();

// add the logging function to the callback list
callbacks.add( foo );

// fire the items on the list, passing an argument
callbacks.fire( 'hello' );
// outputs 'foo: hello'

// lock the callbacks list
callbacks.lock();

// test the lock-state of the list
console.log ( callbacks.locked() ); //true
```

callbacks.empty()

Remove all of the callbacks from a list.

Example

Using `callbacks.empty()` to empty a list of callbacks:

```
// a sample logging function to be added to a callbacks list
var foo = function( value1, value2 ){
  console.log( 'foo:' + value1 + ',' + value2 );
}

// another function to also be added to the list
var bar = function( value1, value2 ){
  console.log( 'bar:' + value1 + ',' + value2 );
}

var callbacks = $.Callbacks();

// add the two functions
callbacks.add( foo );
callbacks.add( bar );

// empty the callbacks list
callbacks.empty();

// check to ensure all callbacks have been removed
console.log( callbacks.has( foo ) ); // false
console.log( callbacks.has( bar ) ); // false
```

callbacks.lock()

Lock a callback list in its current state.

Example

Using `callbacks.lock()` to lock a callback list to avoid further changes being made to the list state:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value);
}

var callbacks = $.Callbacks();

// add the logging function to the callback list
callbacks.add( foo );

// fire the items on the list, passing an argument
callbacks.fire( 'hello' );
// outputs 'foo: hello'

// lock the callbacks list
callbacks.lock();

// try firing the items again
callbacks.fire( 'world' );

// as the list was locked, no items
// were called so 'world' isn't logged
```

callbacks.fire(arguments)
Call all of the callbacks with the given arguments

Arguments
arguments - The argument or list of arguments to pass back to the callback list.

Example

Using `callbacks.fire()` to invoke the callbacks in a list with any arguments that have been passed:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'
callbacks.fire( 'world '); // outputs: 'foo: world'

// add another function to the list
var bar = function( value ){
  console.log( 'bar:' + value );
}

// add this function to the list
callbacks.add( bar );

// fire the items on the list again
callbacks.fire( 'hello again' );
// outputs:
// 'foo: hello again'
// 'bar: hello again'
```

callbacks.remove(callbacks)
Remove a callback or a collection of callbacks from a callback list.

Arguments
callbacks - A function, or array of functions, that are to be removed from the callback list.

Example

Using `callbacks.remove()` to remove callbacks from a callback list:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );
```

```
// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'

// remove 'foo' from the callback list
callbacks.remove( foo );

// fire the items on the list again
callbacks.fire( 'world' );

// nothing output as 'foo' is no longer in the list
```

callbacks.add(callbacks)
Add a callback or a collection of callbacks to a callback list.

Arguments
callbacks - A function, or array of functions, that are to be added to the callback list.

Example

Using `callbacks.add()` to add new callbacks to a callback list:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value );
}

// another function to also be added to the list
var bar = function( value ){
  console.log( 'bar:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' );
// outputs: 'foo: hello'

// add the function 'bar' to the list
callbacks.add( bar );

// fire the items on the list again
callbacks.fire( 'world' );

// outputs:
// 'foo: world'
// 'bar: world'
```

callbacks.disable()
Disable a callback list from doing anything more.

Example

Using `callbacks.disable()` to disable further calls being made to a callback list:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( value );
}

var callbacks = $.Callbacks();

// add the above function to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'foo' ); // outputs: foo

// disable further calls being possible
callbacks.disable();

// attempt to fire with 'foobar' as an argument
callbacks.fire( 'foobar' ); // foobar isn't output
```

callbacks.has(callback)
Determine whether a supplied callback is in a list

Arguments
callback - The callback to search for.

Example

Using `callbacks.has()` to check if a callback list contains a specific callback:

```
// a sample logging function to be added to a callbacks list
var foo = function( value1, value2 ){
  console.log( 'Received:' + value1 + ',' + value2 );
}

// a second function which will not be added to the list
var bar = function( value1, value2 ){
  console.log( 'foobar');
}

var callbacks = $.Callbacks();

// add the log method to the callbacks list
callbacks.add( foo );

// determine which callbacks are in the list

console.log( callbacks.has( foo ) ); // true
console.log( callbacks.has( bar ) ); // false
```

callbacks.fireWith([context], [args])
Call all callbacks in a list with the given context and arguments.

Arguments
context - A reference to the context in which the callbacks in the list should be fired.

args - An argument, or array of arguments, to pass to the callbacks in the list.

Example

Using `callbacks.fireWith()` to fire a list of callbacks with a specific context and an array of arguments:

```
// a sample logging function to be added to a callbacks list
var log = function( value1, value2 ){
  console.log( 'Received:' + value1 + ',' + value2 );
}

var callbacks = $.Callbacks();

// add the log method to the callbacks list
callbacks.add( log );

// fire the callbacks on the list using the context 'window'
// and an arguments array

callbacks.fireWith( window, ['foo','bar']);

// outputs: Received: foo, bar
```

Core

jQuery.holdReady(hold)

Holds or releases the execution of jQuery's ready event.

Arguments

hold - Indicates whether the ready hold is being requested or released

The `$.holdReady()` method allows the caller to delay jQuery's ready event. This *advanced feature* would typically be used by dynamic script loaders that want to load additional JavaScript such as jQuery plugins before allowing the ready event to occur, even though the DOM may be ready. This method must be called early in the document, such as in the `<head>` immediately after the jQuery script tag. Calling this method after the ready event has already fired will have no effect.

To delay the ready event, first call `$.holdReady(true)`. When the ready event should be released to execute, call `$.holdReady(false)`. Note that multiple holds can be put on the ready event, one for each `$.holdReady(true)` call. The ready event will not actually fire until all holds have been released with a corresponding number of `$.holdReady(false)` calls *and* the normal document ready conditions are met. (See ready for more information.)

Example

Delay the ready event until a custom plugin has loaded.

```
$.holdReady(true);
$.getScript("myplugin.js", function() {
   $.holdReady(false);
});
```

jQuery.when(deferreds)

Provides a way to execute callback functions based on one or more objects, usually Deferred objects that represent asynchronous events.

Arguments

deferreds - One or more Deferred objects, or plain JavaScript objects.

If a single Deferred is passed to `jQuery.when`, its Promise object (a subset of the Deferred methods) is returned by the method. Additional methods of the Promise object can be called to attach callbacks, such as deferred.then. When the Deferred is resolved or rejected, usually by the code that created the Deferred originally, the appropriate callbacks will be called. For example, the jqXHR object returned by `jQuery.ajax` is a Deferred and can be used this way:

```
$.when( $.ajax("test.aspx") ).then(function(ajaxArgs){
   alert(ajaxArgs[1]); /* ajaxArgs is [ "success", statusText, jqXHR ] */
});
```

If a single argument is passed to `jQuery.when` and it is not a Deferred, it will be treated as a resolved Deferred and any doneCallbacks attached will be executed immediately. The doneCallbacks are passed the original argument. In this case any failCallbacks you might set are never called since the Deferred is never rejected. For example:

```
$.when( { testing: 123 } ).done(
  function(x){ alert(x.testing); } /* alerts "123" */
);
```

In the case where multiple Deferred objects are passed to `jQuery.when`, the method returns the Promise from a new "master" Deferred object that tracks the aggregate state of all the Deferreds it has been passed. The method will resolve its master Deferred as soon as all the Deferreds resolve, or reject the master Deferred as soon as one of the Deferreds is rejected. If the master Deferred is resolved, it is passed the resolved values of all the Deferreds that were passed to `jQuery.when`. For example, when the Deferreds are `jQuery.ajax()` requests, the arguments will be the jqXHR objects for the requests, in the order they were given in the argument list.

In the multiple-Deferreds case where one of the Deferreds is rejected, `jQuery.when` immediately fires the failCallbacks for its master Deferred. Note that some of the Deferreds may still be unresolved at that point. If you need to perform additional processing for this case, such as canceling any unfinished ajax requests, you can keep references to the underlying jqXHR objects in a closure and inspect/cancel them in the failCallback.

Example

Execute a function after two ajax requests are successful. (See the jQuery.ajax() documentation for a complete description of success and error cases for an ajax request).

```
$.when($.ajax("/page1.php"), $.ajax("/page2.php")).done(function(a1, a2){
  /* a1 and a2 are arguments resolved for the
    page1 and page2 ajax requests, respectively */
  var jqXHR = a1[2]; /* arguments are [ "success", statusText, jqXHR ] */
  if ( /Whip It/.test(jqXHR.responseText) ) {
   alert("First page has 'Whip It' somewhere.");
  }
});
```

Example
Execute the function `myFunc` when both ajax requests are successful, or `myFailure` if either one has an error.

```
$.when($.ajax("/page1.php"), $.ajax("/page2.php"))
 .then(myFunc, myFailure);
```

jQuery.sub()
Creates a new copy of jQuery whose properties and methods can be modified without affecting the original jQuery object.

This method is deprecated as of jQuery 1.7 and will be moved to a plugin in jQuery 1.8.

There are two specific use cases for which jQuery.sub() was created. The first was for providing a painless way of overriding jQuery methods without completely destroying the original methods and another was for helping to do encapsulation and basic namespacing for jQuery plugins.

Note that jQuery.sub() doesn't attempt to do any sort of isolation - that's not its intention. All the methods on the sub'd version of jQuery will still point to the original jQuery (events bound and triggered will still be through the main jQuery, data will be bound to elements through the main jQuery, Ajax queries and events will run through the main jQuery, etc.).

Note that if you're looking to use this for plugin development you should first *strongly* consider using something like the jQuery UI widget factory which manages both state and plugin sub-methods. Some examples of using the jQuery UI widget factory to build a plugin.

The particular use cases of this method can be best described through some examples.

Example
Adding a method to a jQuery sub so that it isn't exposed externally:

```
(function(){
  var sub$ = jQuery.sub();

  sub$.fn.myCustomMethod = function(){
   return 'just for me';
  };

  sub$(document).ready(function() {
   sub$('body').myCustomMethod() // 'just for me'
  });
 })();

 typeof jQuery('body').myCustomMethod // undefined
```

Example
Override some jQuery methods to provide new functionality.

```
(function() {
 var myjQuery = jQuery.sub();

 myjQuery.fn.remove = function() {
  // New functionality: Trigger a remove event
  this.trigger("remove");
```

```
  // Be sure to call the original jQuery remove method
  return jQuery.fn.remove.apply( this, arguments );
 };

 myjQuery(function($) {
  $(".menu").click(function() {
   $(this).find(".submenu").remove();
  });

  // A new remove event is now triggered from this copy of jQuery
  $(document).bind("remove", function(e) {
   $(e.target).parent().hide();
  });
 });
})();

// Regular jQuery doesn't trigger a remove event when removing an element
// This functionality is only contained within the modified 'myjQuery'.
```

Example

Create a plugin that returns plugin-specific methods.

```
(function() {
 // Create a new copy of jQuery using sub()
 var plugin = jQuery.sub();

 // Extend that copy with the new plugin methods
 plugin.fn.extend({
  open: function() {
   return this.show();
  },
  close: function() {
   return this.hide();
  }
 });

 // Add our plugin to the original jQuery
 jQuery.fn.myplugin = function() {
  this.addClass("plugin");

  // Make sure our plugin returns our special plugin version of jQuery
  return plugin( this );
 };
})();

$(document).ready(function() {
 // Call the plugin, open method now exists
 $('#main').myplugin().open();

 // Note: Calling just $("#main").open() won't work as open doesn't exist!
});
```

jQuery.noConflict([removeAll])

Relinquish jQuery's control of the $ variable.

Arguments

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

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 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.

Example

Maps the original object that was referenced by $ back to $.

```
jQuery.noConflict();
// Do something with jQuery
jQuery("div p").hide();
// Do something with another library's $()
$("content").style.display = 'none';
```

Example

Reverts the $ alias and then creates and executes a function to provide the $ as a jQuery alias inside the functions scope. Inside the function the original $ object is not available. This works well for most plugins that don't rely on any other library.

```
jQuery.noConflict();
(function($) {
 $(function() {
  // more code using $ as alias to jQuery
 });
})(jQuery);
// other code using $ as an alias to the other library
```

Example

You can chain the jQuery.noConflict() with the shorthand ready for a compact code.

```
jQuery.noConflict()(function(){
  // code using jQuery
});
// other code using $ as an alias to the other library
```

Example

Creates a different alias instead of jQuery to use in the rest of the script.

```
var j = jQuery.noConflict();
// Do something with jQuery
j("div p").hide();
```

```
// Do something with another library's $()
$("content").style.display = 'none';
```

Example
Completely move jQuery to a new namespace in another object.

```
var dom = {};
dom.query = jQuery.noConflict(true);
```

jQuery(selector, [context])
Accepts a string containing a CSS selector which is then used to match a set of elements.

Arguments
selector - A string containing a selector expression
context - A DOM Element, Document, or jQuery to use as context

In the first formulation listed above, `jQuery()` - which can also be written as `$()` - searches through the DOM for any elements that match the provided selector and creates a new jQuery object that references these elements:

```
$('div.foo');
```

If no elements match the provided selector, the new jQuery object is "empty"; that is, it contains no elements and has `.length` property of 0.

Selector Context

By default, selectors perform their searches within the DOM starting at the document root. However, an alternate context can be given for the search by using the optional second parameter to the `$()` function. For example, to do a search within an event handler, the search can be restricted like so:

```
$('div.foo').click(function() {
 $('span', this).addClass('bar');
});
```

When the search for the span selector is restricted to the context of `this`, only spans within the clicked element will get the additional class.

Internally, selector context is implemented with the `.find()` method, so `$('span', this)` is equivalent to `$(this).find('span')`.

Using DOM elements

The second and third formulations of this function create a jQuery object using one or more DOM elements that were already selected in some other way. A common use of this facility is to call jQuery methods on an element that has been passed to a callback function through the keyword `this`:

```
$('div.foo').click(function() {
 $(this).slideUp();
});
```

This example causes elements to be hidden with a sliding animation when clicked. Because the handler receives the clicked item in the `this` keyword as a bare DOM element, the element must be passed to the `$()` function before applying jQuery methods to it.

XML data returned from an Ajax call can be passed to the `$()` function so individual elements of the XML structure can be retrieved using `.find()` and other DOM traversal methods.

```
$.post('url.xml', function(data) {
 var $child = $(data).find('child');
})
```

Cloning jQuery Objects

When a jQuery object is passed to the `$()` function, a clone of the object is created. This new jQuery object references the same DOM elements as the initial one.

Returning an Empty Set

As of jQuery 1.4, calling the `jQuery()` method with *no arguments* returns an empty jQuery set (with a `.length` property of 0). In previous versions of jQuery, this would return a set containing the document node.

Working With Plain Objects

At present, the only operations supported on plain JavaScript objects wrapped in jQuery are: `.data()`,`.prop()`,`.bind()`, `.unbind()`, `.trigger()` and `.triggerHandler()`. The use of `.data()` (or any method requiring `.data()`) on a plain object will result in a new property on the object called jQuery{randomNumber} (eg. jQuery123456789).

```
// define a plain object
var foo = {foo:'bar', hello:'world'};

// wrap this with jQuery
var $foo = $(foo);

// test accessing property values
var test1 = $foo.prop('foo'); // bar

// test setting property values
$foo.prop('foo', 'foobar');
var test2 = $foo.prop('foo'); // foobar

// test using .data() as summarized above
$foo.data('keyName', 'someValue');
console.log($foo); // will now contain a jQuery{randomNumber} property

// test binding an event name and triggering
$foo.bind('eventName', function (){
    console.log('eventName was called');
});

$foo.trigger('eventName'); // logs 'eventName was called'
```

Should `.trigger('eventName')` be used, it will search for an 'eventName' property on the object and attempt to execute it after any attached jQuery handlers are executed. It does not check whether the property is a function or not. To avoid this behavior, `.triggerHandler('eventName')` should be used instead.

```
$foo.triggerHandler('eventName'); // also logs 'eventName was called'
```

Example

Find all p elements that are children of a div element and apply a border to them.

```
$("div > p").css("border", "1px solid gray");
```

Example

Find all inputs of type radio within the first form in the document.

```
$("input:radio", document.forms[0]);
```

Example
Find all div elements within an XML document from an Ajax response.

```
$("div", xml.responseXML);
```

Example
Set the background color of the page to black.

```
$(document.body).css( "background", "black" );
```

Example
Hide all the input elements within a form.

```
$(myForm.elements).hide()
```

jQuery(html, [ownerDocument])
Creates DOM elements on the fly from the provided string of raw HTML.

Arguments
html - A string of HTML to create on the fly. Note that this parses HTML, **not** XML.
ownerDocument - A document in which the new elements will be created

Creating New Elements

If a string is passed as the parameter to `$()`, jQuery examines the string to see if it looks like HTML (i.e., it has `<tag ... >` somewhere within the string). If not, the string is interpreted as a selector expression, as explained above. But if the string appears to be an HTML snippet, jQuery attempts to create new DOM elements as described by the HTML. Then a jQuery object is created and returned that refers to these elements. You can perform any of the usual jQuery methods on this object:

```
$('<p id="test">My <em>new</em> text</p>').appendTo('body');
```

If the HTML is more complex than a single tag without attributes, as it is in the above example, the actual creation of the elements is handled by the browser's `innerHTML` mechanism. In most cases, jQuery creates a new element and sets the innerHTML property of the element to the HTML snippet that was passed in. When the parameter has a single tag, such as `$('<img />')` or `$('<a></a>')`, jQuery creates the element using the native JavaScript `createElement()` function.

When passing in complex HTML, some browsers may not generate a DOM that exactly replicates the HTML source provided. As mentioned, we use the browser's `.innerHTML` property to parse the passed HTML and insert it into the current document. During this process, some browsers filter out certain elements such as `<html>`, `<title>`, or `<head>` elements. As a result, the elements inserted may not be representative of the original string passed.

Filtering isn't however just limited to these tags. For example, Internet Explorer prior to version 8 will also convert all `href` properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate compatibility layer.

To ensure cross-platform compatibility, the snippet must be well-formed. Tags that can contain other elements should be paired with a closing tag:

```
$('<a href="http://jquery.com"></a>');
```

Alternatively, jQuery allows XML-like tag syntax (with or without a space before the slash):

```
$('<a/>');
```

Tags that cannot contain elements may be quick-closed or not:

```
$('<img />');
$('<input>');
```

When passing HTML to `jQuery()`, please also note that text nodes are not treated as DOM elements. With the exception of a few methods (such as `.content()`), they are generally otherwise ignored or removed. E.g:

```
var el = $('1<br/>2<br/>3'); // returns [<br>, "2", <br>]
el = $('1<br/>2<br/>3 >'); // returns [<br>, "2", <br>, "3 &gt;"]
```

This behaviour is expected.

As of jQuery 1.4, the second argument to `jQuery()` can accept a map consisting of a superset of the properties that can be passed to the .attr() method. Furthermore, any event type can be passed in, and the following jQuery methods can be called: val, css, html, text, data, width, height, or offset. The name `"class"` must be quoted in the map since it is a JavaScript reserved word, and `"className"` cannot be used since it is not the correct attribute name.

Note: Internet Explorer will not allow you to create an `input` or `button` element and change its type; you must specify the type using `'<input type="checkbox" />'` for example. A demonstration of this can be seen below:

Unsupported in IE:

```
$('<input />', {
  type: 'text',
  name: 'test'
}).appendTo("body");
```

Supported workaround:

```
$('<input type="text" />').attr({
  name: 'test'
}).appendTo("body");
```

Example
Create a div element (and all of its contents) dynamically and append it to the body element. Internally, an element is created and its innerHTML property set to the given markup.

```
$("<div><p>Hello</p></div>").appendTo("body")
```

Example
Create some DOM elements.

```
$("<div/>", {
 "class": "test",
 text: "Click me!",
 click: function(){
  $(this).toggleClass("test");
 }
}).appendTo("body");
```

jQuery(callback)
Binds a function to be executed when the DOM has finished loading.

Arguments
callback - The function to execute when the DOM is ready.

This function behaves just like `$(document).ready()`, in that it should be used to wrap other `$()` operations on your page that depend on the DOM being ready. While this function is, technically, chainable, there really isn't much use for chaining against it.

Example
Execute the function when the DOM is ready to be used.

```
$(function(){
```

```
  // Document is ready
 });
```

Example

Use both the shortcut for $(document).ready() and the argument to write failsafe jQuery code using the $ alias, without relying on the global alias.

```
jQuery(function($) {
  // Your code using failsafe $ alias here...
 });
```

CSS

jQuery.cssHooks

Hook directly into jQuery to override how particular CSS properties are retrieved or set, normalize CSS property naming, or create custom properties.

The `$.cssHooks` object provides a way to define functions for getting and setting particular CSS values. It can also be used to create new cssHooks for normalizing CSS3 features such as box shadows and gradients.

For example, some versions of Webkit-based browsers require `-webkit-border-radius` to set the `border-radius` on an element, while earlier Firefox versions require `-moz-border-radius`. A css hook can normalize these vendor-prefixed properties to let `.css()` accept a single, standard property name (`border-radius`, or with DOM property syntax, `borderRadius`).

In addition to providing fine-grained control over how specific style properties are handled, `$.cssHooks` also extends the set of properties available to the `.animate()` method.

Defining a new css hook is straight-forward. The skeleton template below can serve as a guide to creating your own.

```
(function($) {
 // first, check to see if cssHooks are supported
 if ( !$.cssHooks ) {
  // if not, output an error message
  throw("jQuery 1.4.3 or above is required for this plugin to work");
  return;
 }

 $.cssHooks["someCSSProp"] = {
  get: function( elem, computed, extra ) {
   // handle getting the CSS property
  },
  set: function( elem, value ) {
   // handle setting the CSS value
  }
 };
})(jQuery);
```

Feature Testing

Before normalizing a vendor-specific CSS property, first determine whether the browser supports the standard property or a vendor-prefixed variation. For example, to check for support of the `border-radius` property, see if any variation is a member of a temporary element's `style` object.

```
(function($) {
 function styleSupport( prop ) {
  var vendorProp, supportedProp,

    // capitalize first character of the prop to test vendor prefix
    capProp = prop.charAt(0).toUpperCase() + prop.slice(1),
    prefixes = [ "Moz", "Webkit", "O", "ms" ],
    div = document.createElement( "div" );

  if ( prop in div.style ) {

   // browser supports standard CSS property name
   supportedProp = prop;
  } else {

   // otherwise test support for vendor-prefixed property names
   for ( var i = 0; i < prefixes.length; i++ ) {
    vendorProp = prefixes[i] + capProp;
```

```
    if ( vendorProp in div.style ) {
     supportedProp = vendorProp;
     break;
    }
   }
  }

  // avoid memory leak in IE
  div = null;

  // add property to $.support so it can be accessed elsewhere
  $.support[ prop ] = supportedProp;

  return supportedProp;
 }

 // call the function, e.g. testing for "border-radius" support:
 styleSupport( "borderRadius" );
})(jQuery);
```

Defining a complete css hook

To define a complete css hook, combine the support test with a filled-out version of the skeleton template provided in the first example:

```
(function($) {
 if ( !$.cssHooks ) {
  throw("jQuery 1.4.3+ is needed for this plugin to work");
  return;
 }

 function styleSupport( prop ) {
  var vendorProp, supportedProp,
    capProp = prop.charAt(0).toUpperCase() + prop.slice(1),
    prefixes = [ "Moz", "Webkit", "O", "ms" ],
    div = document.createElement( "div" );

  if ( prop in div.style ) {
   supportedProp = prop;
  } else {
   for ( var i = 0; i < prefixes.length; i++ ) {
    vendorProp = prefixes[i] + capProp;
    if ( vendorProp in div.style ) {
     supportedProp = vendorProp;
     break;
    }
   }
  }

  div = null;
  $.support[ prop ] = supportedProp
  return supportedProp;
 }

 var borderRadius = styleSupport( "borderRadius" );

 // Set cssHooks only for browsers that
 // support a vendor-prefixed border radius
 if ( borderRadius && borderRadius !== "borderRadius" ) {
  $.cssHooks.borderRadius = {
   get: function( elem, computed, extra ) {
    return $.css( elem, borderRadius );
```

```
   },
   set: function( elem, value) {
    elem.style[ borderRadius ] = value;
   }
  };
 }
})(jQuery);
```

You can then set the border radius in a supported browser using either the DOM (camelCased) style or the CSS (hyphenated) style:

```
$("#element").css("borderRadius", "10px");
$("#another").css("border-radius", "20px");
```

If the browser lacks support for any form of the CSS property, vendor-prefixed or not, the style is not applied to the element. However, if the browser supports a proprietary alternative, it can be applied to the cssHooks instead.

```
 (function($) {
 // feature test for support of a CSS property
 // and a proprietary alternative
 // ...


 if ( $.support.someCSSProp && $.support.someCSSProp !== "someCSSProp" ) {

  // Set cssHooks for browsers that
  // support only a vendor-prefixed someCSSProp
  $.cssHooks.someCSSProp = {
   get: function( elem, computed, extra ) {
    return $.css( elem, $.support.someCSSProp );
   },
   set: function( elem, value) {
    elem.style[ $.support.someCSSProp ] = value;
   }
  };
 } else if ( supportsProprietaryAlternative ) {
  $.cssHooks.someCSSProp = {
   get: function( elem, computed, extra ) {
    // Handle crazy conversion from the proprietary alternative
   },
   set: function( elem, value ) {
    // Handle crazy conversion to the proprietary alternative
   }
  }
 }

})(jQuery);
```

Special units

By default, jQuery adds a "px" unit to the values passed to the `.css()` method. This behavior can be prevented by adding the property to the `jQuery.cssNumber` object

```
$.cssNumber["someCSSProp"] = true;
```

Animating with cssHooks

A css hook can also hook into jQuery's animation mechanism by adding a property to the `jQuery.fx.step` object:

```
$.fx.step["someCSSProp"] = function(fx){
 $.cssHooks["someCSSProp"].set( fx.elem, fx.now + fx.unit );
};
```

Note that this works best for simple numeric-value animations. More custom code may be required depending on the CSS property, the type of value it returns, and the animation's complexity.

outerWidth([includeMargin])

Get the current computed width for the first element in the set of matched elements, including padding and border.

Arguments

includeMargin - A Boolean indicating whether to include the element's margin in the calculation.

Returns the width of the element, along with left and right padding, border, and optionally margin, in pixels.

If `includeMargin` is omitted or `false`, the padding and border are included in the calculation; if `true`, the margin is also included.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.

Example

Get the outerWidth of a paragraph.

```
var p = $("p:first");
$("p:last").text( "outerWidth:" + p.outerWidth()+ " , outerWidth(true):" + p.outerWidth(true) );
```

outerHeight([includeMargin])

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

Arguments

includeMargin - A Boolean indicating whether to include the element's margin in the calculation.

The top and bottom padding and border are always included in the `.outerHeight()` calculation; if the `includeMargin` argument is set to `true`, the margin (top and bottom) is also included.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.

Example

Get the outerHeight of a paragraph.

```
var p = $("p:first");
$("p:last").text( "outerHeight:" + p.outerHeight() + " , outerHeight(true):" + p.outerHeight(true) );
```

innerWidth()

Get the current computed width for the first element in the set of matched elements, including padding but not border.

This method returns the width of the element, including left and right padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.

Example
Get the innerWidth of a paragraph.

```
var p = $("p:first");
$("p:last").text( "innerWidth:" + p.innerWidth() );
```

innerHeight()
Get the current computed height for the first element in the set of matched elements, including padding but not border.

This method returns the height of the element, including top and bottom padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.

Example
Get the innerHeight of a paragraph.

```
var p = $("p:first");
$("p:last").text( "innerHeight:" + p.innerHeight() );
```

width()
Get the current computed width for the first element in the set of matched elements.

The difference between `.css(width)` and `.width()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.width()` method is recommended when an element's width needs to be used in a mathematical calculation.

This method is also able to find the width of the window and document.

```
$(window).width();  // returns width of browser viewport
$(document).width(); // returns width of HTML document
```

Note that `.width()` will always return the content width, regardless of the value of the CSS `box-sizing` property.

Example
Show various widths. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showWidth(ele, w) {
 $("div").text("The width for the " + ele +
        " is " + w + "px.");
}
$("#getp").click(function () {
 showWidth("paragraph", $("p").width());
});
$("#getd").click(function () {
 showWidth("document", $(document).width());
});
$("#getw").click(function () {
 showWidth("window", $(window).width());
});
```

width(value)
Set the CSS width of each element in the set of matched elements.

Arguments

value - An integer representing the number of pixels, or an integer along with an optional unit of measure appended (as a string).

When calling `.width("value")`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, any valid CSS measurement may be used for the width (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS width property does not include padding, border, or margin, unless the `box-sizing` CSS property is used.

If no explicit unit is specified (like "em" or "%") then "px" is assumed.

Note that `.width("value")` sets the width of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the `outerWidth` of the box instead of the content width.

Example
Change the width of each div the first time it is clicked (and change its color).

```
(function() {
 var modWidth = 50;
 $("div").one('click', function () {
  $(this).width(modWidth).addClass("mod");
 modWidth -= 8;
 });
})();
```

height()
Get the current computed height for the first element in the set of matched elements.

The difference between `.css('height')` and `.height()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.height()` method is recommended when an element's height needs to be used in a mathematical calculation.

This method is also able to find the height of the window and document.

```
$(window).height();  // returns height of browser viewport
$(document).height(); // returns height of HTML document
```

Note that `.height()` will always return the content height, regardless of the value of the CSS `box-sizing` property.

Note: Although `style` and `script` tags will report a value for `.width()` or `height()` when absolutely positioned and given `display:block`, it is strongly discouraged to call those methods on these tags. In addition to being a bad practice, the results may also prove unreliable.

Example
Show various heights. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showHeight(ele, h) {
   $("div").text("The height for the " + ele +
          " is " + h + "px.");
  }
  $("#getp").click(function () {
   showHeight("paragraph", $("p").height());
  });
  $("#getd").click(function () {
   showHeight("document", $(document).height());
  });
  $("#getw").click(function () {
   showHeight("window", $(window).height());
  });
```

height(value)
Set the CSS height of every matched element.

Arguments
value - An integer representing the number of pixels, or an integer with an optional unit of measure appended (as a string).

When calling `.height(value)`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, a valid CSS measurement must be provided for the height (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS height property does not include padding, border, or margin.

If no explicit unit was specified (like 'em' or '%') then "px" is concatenated to the value.

Note that `.height(value)` sets the height of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the outerHeight of the box instead of the content height.

Example
To set the height of each div on click to 30px plus a color change.

```
$("div").one('click', function () {
   $(this).height(30)
       .css({cursor:"auto", backgroundColor:"green"});
  });
```

scrollLeft()
Get the current horizontal position of the scroll bar for the first element in the set of matched elements.

The horizontal scroll position is the same as the number of pixels that are hidden from view to the left of the scrollable area. If the scroll bar is at the very left, or if the element is not scrollable, this number will be `0`.

Note: `.scrollLeft()`, when called directly or animated as a property using `.animate()`, will not work if the element it is being applied to is hidden.

Example
Get the scrollLeft of a paragraph.

```
var p = $("p:first");
$("p:last").text( "scrollLeft:" + p.scrollLeft() );
```

scrollLeft(value)
Set the current horizontal position of the scroll bar for each of the set of matched elements.

Arguments
value - An integer indicating the new position to set the scroll bar to.

The horizontal scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollLeft` positions the horizontal scroll of each matched element.

Example
Set the scrollLeft of a div.

```
$("div.demo").scrollLeft(300);
```

scrollTop()
Get the current vertical position of the scroll bar for the first element in the set of matched elements.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. If the scroll bar is at the very top, or if the element is not scrollable, this number will be `0`.

Example
Get the scrollTop of a paragraph.

```
var p = $("p:first");
```

```
$("p:last").text( "scrollTop:" + p.scrollTop() );
```

scrollTop(value)
Set the current vertical position of the scroll bar for each of the set of matched elements.

Arguments
value - An integer indicating the new position to set the scroll bar to.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollTop` positions the vertical scroll of each matched element.

Example
Set the scrollTop of a div.

```
$("div.demo").scrollTop(300);
```

position()
Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

The `.position()` method allows us to retrieve the current position of an element *relative to the offset parent*. Contrast this with `.offset()`, which retrieves the current position *relative to the document*. When positioning a new element near another one and within the same containing DOM element, `.position()` is the more useful.

Returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the position coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.

Example
Access the position of the second paragraph:

```
var p = $("p:first");
var position = p.position();
$("p:last").text( "left: " + position.left + ", top: " + position.top );
```

offset()
Get the current coordinates of the first element in the set of matched elements, relative to the document.

The `.offset()` method allows us to retrieve the current position of an element *relative to the document*. Contrast this with `.position()`, which retrieves the current position *relative to the offset parent*. When positioning a new element on top of an existing one for global manipulation (in particular, for implementing drag-and-drop), `.offset()` is the more useful.

`.offset()` returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the offset coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.

While it is possible to get the coordinates of elements with `visibility:hidden` set, `display:none` is excluded from the rendering tree and thus has a position that is undefined.

Example
Access the offset of the second paragraph:

```
var p = $("p:last");
var offset = p.offset();
p.html( "left: " + offset.left + ", top: " + offset.top );
```

Example

Click to see the offset.

```
$("*", document.body).click(function (e) {
 var offset = $(this).offset();
 e.stopPropagation();
 $("#result").text(this.tagName + " coords ( " + offset.left + ", " +
                 offset.top + " )");
});
```

offset(coordinates)

Set the current coordinates of every element in the set of matched elements, relative to the document.

Arguments

coordinates - An object containing the properties `top` and `left`, which are integers indicating the new top and left coordinates for the elements.

The `.offset()` setter method allows us to reposition an element. The element's position is specified *relative to the document*. If the element's `position` style property is currently `static`, it will be set to `relative` to allow for this repositioning.

Example

Set the offset of the second paragraph:

```
$("p:last").offset({ top: 10, left: 30 });
```

css(propertyName)

Get the value of a style property for the first element in the set of matched elements.

Arguments

propertyName - A CSS property.

The `.css()` method is a convenient way to get a style property from the first matched element, especially in light of the different ways browsers access most of those properties (the `getComputedStyle()` method in standards-based browsers versus the `currentStyle` and `runtimeStyle` properties in Internet Explorer) and the different terms browsers use for certain properties. For example, Internet Explorer's DOM implementation refers to the `float` property as `styleFloat`, while W3C standards-compliant browsers refer to it as `cssFloat`. The `.css()` method accounts for such differences, producing the same result no matter which term we use. For example, an element that is floated left will return the string `left` for each of the following three lines:

- `$('div.left').css('float');`
- `$('div.left').css('cssFloat');`
- `$('div.left').css('styleFloat');`

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css('background-color')` and `.css('backgroundColor')`. Different browsers may return CSS color values that are logically but not textually equal, e.g., #FFF, #ffffff, and rgb(255,255,255).

Shorthand CSS properties (e.g. margin, background, border) are not supported. For example, if you want to retrieve the rendered margin, use: `$(elem).css('marginTop')` and `$(elem).css('marginRight')`, and so on.

Example

To access the background color of a clicked div.

```
$("div").click(function () {
 var color = $(this).css("background-color");
 $("#result").html("That div is <span style='color:" +
          color + ";'>" + color + "</span>.");
});
```

css(propertyName, value)

Set one or more CSS properties for the set of matched elements.

Arguments

propertyName - A CSS property name.

value - A value to set for the property.

As with the `.prop()` method, the `.css()` method makes setting properties of elements quick and easy. This method can take either a property name and value as separate parameters, or a single map of key-value pairs (JavaScript object notation).

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css({'background-color': '#ffe', 'border-left': '5px solid #ccc'})` and `.css({backgroundColor: '#ffe', borderLeft: '5px solid #ccc'})`. Notice that with the DOM notation, quotation marks around the property names are optional, but with CSS notation they're required due to the hyphen in the name.

When using `.css()` as a setter, jQuery modifies the element's `style` property. For example, `$('#mydiv').css('color', 'green')` is equivalent to `document.getElementById('mydiv').style.color = 'green'`. Setting the value of a style property to an empty string - e.g. `$('#mydiv').css('color', '')` - removes that property from an element if it has already been directly applied, whether in the HTML style attribute, through jQuery's `.css()` method, or through direct DOM manipulation of the `style` property. It does not, however, remove a style that has been applied with a CSS rule in a stylesheet or `<style>` element.

As of jQuery 1.6, `.css()` accepts relative values similar to `.animate()`. Relative values are a string starting with `+=` or `-=` to increment or decrement the current value. For example, if an element's padding-left was 10px, `.css( "padding-left", "+=15" )` would result in a total padding-left of 25px.

As of jQuery 1.4, `.css()` allows us to pass a function as the property value:

```
$('div.example').css('width', function(index) {
 return index * 50;
});
```

This example sets the widths of the matched elements to incrementally larger values.

Note: If nothing is returned in the setter function (ie. `function(index, style){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example

To change the color of any paragraph to red on mouseover event.

```
$("p").mouseover(function () {
  $(this).css("color","red");
 });
```

Example

Increase the width of #box by 200 pixels

```
$("#box").one( "click", function () {
  $( this ).css( "width","+=200" );
 });
```

Example

To highlight a clicked word in the paragraph.

```
var words = $("p:first").text().split(" ");
 var text = words.join("</span> <span>");
 $("p:first").html("<span>" + text + "</span>");
 $("span").click(function () {
  $(this).css("background-color","yellow");
 });
```

Example

To set the color of all paragraphs to red and background to blue:

```
$("p").hover(function () {
  $(this).css({'background-color' : 'yellow', 'font-weight' : 'bolder'});
 }, function () {
  var cssObj = {
   'background-color' : '#ddd',
```

```
   'font-weight' : '',
   'color' : 'rgb(0,40,244)'
  }
  $(this).css(cssObj);
 });
```

Example

Increase the size of a div when you click it:

```
$("div").click(function() {
  $(this).css({
   width: function(index, value) {
    return parseFloat(value) * 1.2;
   },
   height: function(index, value) {
    return parseFloat(value) * 1.2;
   }

  });
 });
```

toggleClass(className)

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Arguments

className - One or more class names (separated by spaces) to be toggled for each element in the matched set.

This method takes one or more class names as its parameter. In the first version, if an element in the matched set of elements already has the class, then it is removed; if an element does not have the class, then it is added. For example, we can apply `.toggleClass()` to a simple `<div>`:

```
<div class="tumble">Some text.</div>
```

The first time we apply `$('div.tumble').toggleClass('bounce')`, we get the following:

```
<div class="tumble bounce">Some text.</div>
```

The second time we apply `$('div.tumble').toggleClass('bounce')`, the `<div>` class is returned to the single `tumble` value:

```
<div class="tumble">Some text.</div>
```

Applying `.toggleClass('bounce spin')` to the same `<div>` alternates between `<div class="tumble bounce spin">` and `<div class="tumble">`.

The second version of `.toggleClass()` uses the second parameter for determining whether the class should be added or removed. If this parameter's value is `true`, then the class is added; if `false`, the class is removed. In essence, the statement:

```
$('#foo').toggleClass(className, addOrRemove);
```

is equivalent to:

```
if (addOrRemove) {
  $('#foo').addClass(className);
 }
 else {
  $('#foo').removeClass(className);
 }
```

As of jQuery 1.4, if no arguments are passed to `.toggleClass()`, all class names on the element the first time `.toggleClass()` is called will be

toggled. Also as of jQuery 1.4, the class name to be toggled can be determined by passing in a function.

```
$('div.foo').toggleClass(function() {
 if ($(this).parent().is('.bar')) {
  return 'happy';
 } else {
  return 'sad';
 }
});
```

This example will toggle the `happy` class for `<div class="foo">` elements if their parent element has a class of `bar`; otherwise, it will toggle the `sad` class.

Example
Toggle the class 'highlight' when a paragraph is clicked.

```
$("p").click(function () {
   $(this).toggleClass("highlight");
  });
```

Example
Add the "highlight" class to the clicked paragraph on every third click of that paragraph, remove it every first and second click.

```
var count = 0;
$("p").each(function() {
 var $thisParagraph = $(this);
 var count = 0;
 $thisParagraph.click(function() {
  count++;
  $thisParagraph.find("span").text('clicks: ' + count);
  $thisParagraph.toggleClass("highlight", count % 3 == 0);
 });
});
```

Example
Toggle the class name(s) indicated on the buttons for each div.

```
var cls = ['', 'a', 'a b', 'a b c'];
var divs = $('div.wrap').children();
var appendClass = function() {
 divs.append(function() {
  return '<div>' + (this.className || 'none') + '</div>';
 });
};

appendClass();

$('button').bind('click', function() {
 var tc = this.className || undefined;
 divs.toggleClass(tc);
 appendClass();
});

$('a').bind('click', function(event) {
 event.preventDefault();
 divs.empty().each(function(i) {
  this.className = cls[i];
 });
 appendClass();
});
```

removeClass([className])

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Arguments
className - One or more space-separated classes to be removed from the class attribute of each matched element.

If a class name is included as a parameter, then only that class will be removed from the set of matched elements. If no class names are specified in the parameter, all classes will be removed.

More than one class may be removed at a time, separated by a space, from the set of matched elements, like so:

```
$('p').removeClass('myClass yourClass')
```

This method is often used with `.addClass()` to switch elements' classes from one to another, like so:

```
$('p').removeClass('myClass noClass').addClass('yourClass');
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

To replace all existing classes with another class, we can use `.attr('class', 'newClass')` instead.

As of jQuery 1.4, the `.removeClass()` method allows us to indicate the class to be removed by passing in a function.

```
$('li:last').removeClass(function() {
     return $(this).prev().attr('class');
    });
```

This example removes the class name of the penultimate `<li>` from the last `<li>`.

Example
Remove the class 'blue' from the matched elements.

```
$("p:even").removeClass("blue");
```

Example
Remove the class 'blue' and 'under' from the matched elements.

```
$("p:odd").removeClass("blue under");
```

Example
Remove all the classes from the matched elements.

```
$("p:eq(1)").removeClass();
```

hasClass(className)
Determine whether any of the matched elements are assigned the given class.

Arguments
className - The class name to search for.

Elements may have more than one class assigned to them. In HTML, this is represented by separating the class names with a space:

```
<div id="mydiv" class="foo bar"></div>
```

The `.hasClass()` method will return `true` if the class is assigned to an element, even if other classes also are. For example, given the HTML above, the following will return `true`:

```
$('#mydiv').hasClass('foo')
```

As would:

```
$('#mydiv').hasClass('bar')
```

While this would return `false`:

```
$('#mydiv').hasClass('quux')
```

Example
Looks for the paragraph that contains 'selected' as a class.

```
$("div#result1").append($("p:first").hasClass("selected").toString());
$("div#result2").append($("p:last").hasClass("selected").toString());
$("div#result3").append($("p").hasClass("selected").toString());
```

addClass(className)
Adds the specified class(es) to each of the set of matched elements.

Arguments
className - One or more class names to be added to the class attribute of each matched element.

It's important to note that this method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.

More than one class may be added at a time, separated by a space, to the set of matched elements, like so:

```
$("p").addClass("myClass yourClass");
```

This method is often used with `.removeClass()` to switch elements' classes from one to another, like so:

```
$("p").removeClass("myClass noClass").addClass("yourClass");
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

As of jQuery 1.4, the `.addClass()` method's argument can receive a function.

```
$("ul li:last").addClass(function(index) {
 return "item-" + index;
});
```

Given an unordered list with five `<li>` elements, this example adds the class "item-4" to the last `<li>`.

Example
Adds the class "selected" to the matched elements.

```
$("p:last").addClass("selected");
```

Example
Adds the classes "selected" and "highlight" to the matched elements.

```
$("p:last").addClass("selected highlight");
```

Example
Pass in a function to `.addClass()` to add the "green" class to a div that already has a "red" class.

```
$("div").addClass(function(index, currentClass) {
  var addedClass;

  if ( currentClass === "red" ) {
   addedClass = "green";
   $("p").text("There is one green div");
  }

  return addedClass;
```

```
});
```

Data

jQuery.hasData(element)

Determine whether an element has any jQuery data associated with it.

Arguments

element - A DOM element to be checked for data.

The `jQuery.hasData()` method provides a way to determine if an element currently has any values that were set using `jQuery.data()`. If no data is associated with an element (there is no data object at all or the data object is empty), the method returns `false`; otherwise it returns `true`.

The primary advantage of `jQuery.hasData(element)` is that it does not create and associate a data object with the element if none currently exists. In contrast, `jQuery.data(element)` always returns a data object to the caller, creating one if no data object previously existed.

Note that jQuery's event system uses the jQuery data API to store event handlers. Therefore, binding an event to an element using `.on()`, `.bind()`, `.live()`, `.delegate()`, or one of the shorthand event methods also associates a data object with that element.

Example

Set data on an element and see the results of hasData.

```
var $p = jQuery("p"), p = $p[0];
$p.append(jQuery.hasData(p)+" "); /* false */

$.data(p, "testing", 123);
$p.append(jQuery.hasData(p)+" "); /* true*/

$.removeData(p, "testing");
$p.append(jQuery.hasData(p)+" "); /* false */

$p.on('click', function() {});
$p.append(jQuery.hasData(p)+" "); /* true */

$p.off('click');
$p.append(jQuery.hasData(p)+" "); /* false */
```

jQuery.removeData(element, [name])

Remove a previously-stored piece of data.

Arguments

element - A DOM element from which to remove data.
name - A string naming the piece of data to remove.

Note: This is a low-level method, you should probably use `.removeData()` instead.

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

Example

Set a data store for 2 names then remove one of them.

```
var div = $("div")[0];
$("span:eq(0)").text("" + $("div").data("test1"));
jQuery.data(div, "test1", "VALUE-1");
jQuery.data(div, "test2", "VALUE-2");
$("span:eq(1)").text("" + jQuery.data(div, "test1"));
jQuery.removeData(div, "test1");
$("span:eq(2)").text("" + jQuery.data(div, "test1"));
$("span:eq(3)").text("" + jQuery.data(div, "test2"));
```

jQuery.data(element, key, value)

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

Arguments

element - The DOM element to associate with the data.
key - A string naming the piece of data to set.
value - The new data value.

Note: This is a low-level method; a more convenient `.data()` is also available.

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');
```

Note: 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.

Example

Store then retrieve a value from the div element.

```
var div = $("div")[0];
  jQuery.data(div, "test", { first: 16, last: "pizza!" });
  $("span:first").text(jQuery.data(div, "test").first);
  $("span:last").text(jQuery.data(div, "test").last);
```

jQuery.data(element, key)

Returns value at named data store for the element, as set by `jQuery.data(element, name, value)`, or the full data store for the element.

Arguments

element - The DOM element to query for the data.
key - Name of the data stored.

Note: This is a low-level method; a more convenient `.data()` is also available.

Regarding HTML5 data-* attributes: This low-level method does NOT retrieve the `data-*` attributes unless the more convenient `.data()` method has already retrieved them.

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 from memory leaks. We can retrieve several distinct values for a single element one at a time, or as a set:

```
alert(jQuery.data( document.body, 'foo' ));
alert(jQuery.data( document.body ));
```

The above lines alert the data values that were set on the `body` element. If nothing was set on that element, an empty string is returned.

Calling `jQuery.data(element)` retrieves all of the element's associated values as a JavaScript object. Note that jQuery itself uses this method to store data for internal use, such as event handlers, so do not assume that it contains only data that your own code has stored.

Note: 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.

Example

Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value, div = $("div")[0];
```

```
 switch ($("button").index(this)) {
  case 0 :
   value = jQuery.data(div, "blah");
   break;
  case 1 :
   jQuery.data(div, "blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   jQuery.data(div, "blah", 86);
   value = "Stored!";
   break;
  case 3 :
   jQuery.removeData(div, "blah");
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

jQuery.dequeue(element, [queueName])

Execute the next function on the queue for the matched element.

Arguments

element - A DOM element from which to remove and execute a queued function.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Note: This is a low-level method, you should probably use `.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.

Example

Use jQuery.dequeue() to end a custom queue function which allows the queue to keep going.

```
$("button").click(function () {
   $("div").animate({left:'+=200px'}, 2000);
   $("div").animate({top:'0px'}, 600);
   $("div").queue(function () {
    $(this).toggleClass("red");
     $.dequeue( this );
       });
   $("div").animate({left:'10px', top:'30px'}, 700);
  });
```

jQuery.queue(element, [queueName])

Show the queue of functions to be executed on the matched element.

Arguments

element - A DOM element to inspect for an attached queue.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Note: This is a low-level method, you should probably use `.queue()` instead.

Example

Show the length of the queue.

```
$("#show").click(function () {
   var n = jQuery.queue( $("div")[0], "fx" );
   $("span").text("Queue length is: " + n.length);
```

```
});
function runIt() {
 $("div").show("slow");
 $("div").animate({left:'+=200'},2000);
 $("div").slideToggle(1000);
 $("div").slideToggle("fast");
 $("div").animate({left:'-=200'},1500);
 $("div").hide("slow");
 $("div").show(1200);
 $("div").slideUp("normal", runIt);
}
runIt();
```

jQuery.queue(element, queueName, newQueue)

Manipulate the queue of functions to be executed on the matched element.

Arguments

element - A DOM element where the array of queued functions is attached.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.
newQueue - An array of functions to replace the current queue contents.

Note: This is a low-level method, you should probably use `.queue()` instead.

Every element can have one or more queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution.

The `jQuery.queue()` method allows us to directly manipulate this queue of functions. Calling `jQuery.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue.

Note that when adding a function with `jQuery.queue()`, we should ensure that `jQuery.dequeue()` is eventually called so that the next function in line executes.

Example

Queue a custom function.

```
$(document.body).click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).addClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").animate({left:'-=200'},500);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).removeClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").slideUp();
  });
```

Example

Set a queue array to delete the queue.

```
$("#start").click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},5000);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).addClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").animate({left:'-=200'},1500);
   jQuery.queue( $("div")[0], "fx", function () {
```

```
    $(this).removeClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").slideUp();
  });
  $("#stop").click(function () {
   jQuery.queue( $("div")[0], "fx", [] );
   $("div").stop();
  });
```

clearQueue([queueName])

Remove from the queue all items that have not yet been run.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

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.

Example

Empty the queue.

```
$("#start").click(function () {

 var myDiv = $("div");
 myDiv.show("slow");
 myDiv.animate({left:'+=200'},5000);
 myDiv.queue(function () {
  var _this = $(this);
  _this.addClass("newcolor");
  _this.dequeue();
 });

 myDiv.animate({left:'-=200'},1500);
 myDiv.queue(function () {
  var _this = $(this);
  _this.removeClass("newcolor");
  _this.dequeue();
 });
 myDiv.slideUp();

});

$("#stop").click(function () {
 var myDiv = $("div");
 myDiv.clearQueue();
 myDiv.stop();
});
```

removeData([name])

Remove a previously-stored piece of data.

Arguments

name - A string naming the piece of data to delete.

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. Removing data from jQuery's internal `.data()` cache does not effect any HTML5 `data-` attributes in a document; use `.removeAttr()` to remove those.

When using `.removeData("name")`, jQuery will attempt to locate a `data-` attribute on the element if no property by that name is in the internal data cache. To avoid a re-query of the `data-` attribute, set the name to a value of either `null` or `undefined` (e.g. `.data("name", undefined)`) rather than using `.removeData()`.

As of jQuery 1.7, when called with an array of keys or a string of space-separated keys, `.removeData()` deletes the value of each key in that array or string.

As of jQuery 1.4.3, calling `.removeData()` will cause the value of the property being removed to revert to the value of the data attribute of the same name in the DOM, rather than being set to `undefined`.

Example
Set a data store for 2 names then remove one of them.

```
$("span:eq(0)").text("" + $("div").data("test1"));
  $("div").data("test1", "VALUE-1");
  $("div").data("test2", "VALUE-2");
  $("span:eq(1)").text("" + $("div").data("test1"));
  $("div").removeData("test1");
  $("span:eq(2)").text("" + $("div").data("test1"));
  $("span:eq(3)").text("" + $("div").data("test2"));
```

data(key, value)
Store arbitrary data associated with the matched elements.

Arguments
key - A string naming the piece of data to set.
value - The new data value; it can be any Javascript type including Array or Object.

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 later:

```
$('body').data('foo', 52);
$('body').data('bar', { myType: 'test', count: 40 });

$('body').data('foo'); // 52
$('body').data(); // {foo: 52, bar: { myType: 'test', count: 40 }}
```

In jQuery 1.4.3 setting an element's data object with `.data(obj)` extends the data previously stored with that element. jQuery itself uses the `.data()` method to save information under the names 'events' and 'handle', and also reserves any data name starting with an underscore ('_') for internal use.

Prior to jQuery 1.4.3 (starting in jQuery 1.4) the .data() method completely replaced all data, instead of just extending the data object. If you are using third-party plugins it may not be advisable to completely replace the element's data object, since plugins may have also set data.

Due to the way browsers interact with plugins and external code, the `.data()` method cannot be used on `<object>` (unless it's a Flash plugin), `<applet>` or `<embed>` elements.

Example
Store then retrieve a value from the div element.

```
$("div").data("test", { first: 16, last: "pizza!" });
$("span:first").text($("div").data("test").first);
$("span:last").text($("div").data("test").last);
```

data(key)
Returns value at named data store for the first element in the jQuery collection, as set by data(name, value).

Arguments
key - Name of the data stored.

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 retrieve several distinct values for a single element one at a time, or as a set:

```
alert($('body').data('foo'));
alert($('body').data());
```

The above lines alert the data values that were set on the `body` element. If no data at all was set on that element, `undefined` is returned.

```
alert( $("body").data("foo")); //undefined
$("body").data("bar", "foobar");
alert( $("body").data("bar")); //foobar
```

HTML5 data-* Attributes

As of jQuery 1.4.3 HTML 5 data- attributes will be automatically pulled in to jQuery's data object. The treatment of attributes with embedded dashes was changed in jQuery 1.6 to conform to the W3C HTML5 specification.

For example, given the following HTML:

```
<div data-role="page" data-last-value="43" data-hidden="true" data-options='{"name":"John"}'></div>
```

All of the following jQuery code will work.

```
$("div").data("role") === "page";
$("div").data("lastValue") === 43;
$("div").data("hidden") === true;
$("div").data("options").name === "John";
```

Every attempt is made to convert the string to a JavaScript value (this includes booleans, numbers, objects, arrays, and null) otherwise it is left as a string. To retrieve the value's attribute as a string without any attempt to convert it, use the `attr()` method. When the data attribute is an object (starts with '{') or array (starts with '[') then `jQuery.parseJSON` is used to parse the string; it must follow valid JSON syntax*including quoted property names*. The data- attributes are pulled in the first time the data property is accessed and then are no longer accessed or mutated (all data values are then stored internally in jQuery).

Calling `.data()` with no parameters retrieves all of the values as a JavaScript object. This object can be safely cached in a variable as long as a new object is not set with `.data(obj)`. Using the object directly to get or set values is faster than making individual calls to `.data()` to get or set each value:

```
var mydata = $("#mydiv").data();
if ( mydata.count < 9 ) {
  mydata.count = 43;
  mydata.status = "embiggened";
}
```

Example
Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value;

 switch ($("button").index(this)) {
  case 0 :
   value = $("div").data("blah");
```

```
   break;
  case 1 :
   $("div").data("blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   $("div").data("blah", 86);
   value = "Stored!";
   break;
  case 3 :
   $("div").removeData("blah");
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

dequeue([queueName])

Execute the next function on the queue for the matched elements.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

When `.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 `.dequeue()` to be called, so that the sequence can continue.

Example

Use dequeue to end a custom queue function which allows the queue to keep going.

```
$("button").click(function () {
 $("div").animate({left:'+=200px'}, 2000);
 $("div").animate({top:'0px'}, 600);
 $("div").queue(function () {
  $(this).toggleClass("red");
  $(this).dequeue();
 });
 $("div").animate({left:'10px', top:'30px'}, 700);
});
```

queue([queueName])

Show the queue of functions to be executed on the matched elements.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Example

Show the length of the queue.

```
var div = $("div");

function runIt() {
 div.show("slow");
 div.animate({left:'+=200'},2000);
 div.slideToggle(1000);
 div.slideToggle("fast");
 div.animate({left:'-=200'},1500);
 div.hide("slow");
 div.show(1200);
 div.slideUp("normal", runIt);
}
```

```
function showIt() {
 var n = div.queue("fx");
 $("span").text( n.length );
 setTimeout(showIt, 100);
}

runIt();
showIt();
```

queue([queueName], newQueue)

Manipulate the queue of functions to be executed on the matched elements.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.
newQueue - An array of functions to replace the current queue contents.

Every element can have one to many queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution. The typical example of this is calling multiple animation methods on an element. For example:

```
$('#foo').slideUp().fadeIn();
```

When this statement is executed, the element begins its sliding animation immediately, but the fading transition is placed on the `fx` queue to be called only once the sliding transition is complete.

The `.queue()` method allows us to directly manipulate this queue of functions. Calling `.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue.

This feature is similar to providing a callback function with an animation method, but does not require the callback to be given at the time the animation is performed.

```
$('#foo').slideUp();
$('#foo').queue(function() {
 alert('Animation complete.');
 $(this).dequeue();
});
```

This is equivalent to:

```
$('#foo').slideUp(function() {
 alert('Animation complete.');
});
```

Note that when adding a function with `.queue()`, we should ensure that `.dequeue()` is eventually called so that the next function in line executes.

As of jQuery 1.4, the function that's called is passed another function as the first argument. When called, this automatically dequeues the next item and keeps the queue moving. We use it as follows:

```
$("#test").queue(function(next) {
  // Do some stuff...
  next();
});
```

Example

Queue a custom function.

```
$(document.body).click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   $("div").queue(function () {
    $(this).addClass("newcolor");
```

```
    $(this).dequeue();
   });
   $("div").animate({left:'-=200'},500);
   $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
   });
   $("div").slideUp();
  });
```

Example

Set a queue array to delete the queue.

```
$("#start").click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},5000);
   $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
   });
   $("div").animate({left:'-=200'},1500);
   $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
   });
   $("div").slideUp();
  });
  $("#stop").click(function () {
   $("div").queue("fx", []);
   $("div").stop();
  });
```

Deferred Object

deferred.progress(progressCallbacks)
Add handlers to be called when the Deferred object generates progress notifications.

Arguments
progressCallbacks - A function, or array of functions, that is called when the Deferred generates progress notifications.

The argument can be either a single function or an array of functions. When the Deferred generates progress notifications by calling `notify` or `notifyWith`, the `progressCallbacks` are called. Since `deferred.progress()` returns the Deferred object, other methods of the Deferred object can be chained to this one. When the Deferred is resolved or rejected, progress callbacks will no longer be called. For more information, see the documentation for Deferred object.

deferred.notifyWith(context, [args])
Call the progressCallbacks on a Deferred object with the given context and `args`.

Arguments
context - Context passed to the progressCallbacks as the `this` object.
args - Optional arguments that are passed to the progressCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state or reporting status by returning a restricted Promise object through deferred.promise().

When `deferred.notifyWith` is called, any progressCallbacks added by deferred.then or deferred.progress are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.notifyWith()`. Any calls to `.notifyWith()` after a Deferred is resolved or rejected (or any progressCallbacks added after that) are ignored. For more information, see the documentation for Deferred object.

deferred.notify(args)
Call the progressCallbacks on a Deferred object with the given `args`.

Arguments
args - Optional arguments that are passed to the progressCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state or reporting status by returning a restricted Promise object through deferred.promise().

When `deferred.notify` is called, any progressCallbacks added by deferred.then or deferred.progress are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.notify()`. Any calls to `.notify()` after a Deferred is resolved or rejected (or any progressCallbacks added after that) are ignored. For more information, see the documentation for Deferred object.

deferred.state()
Determine the current state of a Deferred object.

The deferred.state() method returns a string representing the current state of the Deferred object. The Deferred object can be in one of three states:
- **"pending"**: The Deferred object is not yet in a completed state (neither "rejected" nor "resolved").
- **"resolved"**: The Deferred object is in the resolved state, meaning that either deferred.resolve() or deferred.resolveWith() has been called for the object and the doneCallbacks have been called (or are in the process of being called).
- **"rejected"**: The Deferred object is in the rejected state, meaning that either deferred.reject() or deferred.rejectWith() has been called for the object and the failCallbacks have been called (or are in the process of being called).

This method is primarily useful for debugging to determine, for example, whether a Deferred has already been resolved even though you are inside code that intended to reject it.

deferred.pipe([doneFilter], [failFilter])
Utility method to filter and/or chain Deferreds.

Arguments
doneFilter - An optional function that is called when the Deferred is resolved.
failFilter - An optional function that is called when the Deferred is rejected.

The `deferred.pipe()` method returns a new promise that filters the status and values of a deferred through a function. The `doneFilter` and `failFilter` functions filter the original deferred's resolved / rejected status and values. **As of jQuery 1.7**, the method also accepts a `progressFilter` function to filter any calls to the original deferred's `notify` or `notifyWith` methods. These filter functions can return a new value to be passed along to the piped promise's `done()` or `fail()` callbacks, or they can return another observable object (Deferred, Promise, etc) which will pass its resolved / rejected status and values to the piped promise's callbacks. If the filter function used is `null`, or not specified, the piped promise will be resolved or rejected with the same values as the original.

Example
Filter resolve value:

```
var defer = $.Deferred(),
  filtered = defer.pipe(function( value ) {
   return value * 2;
  });

defer.resolve( 5 );
filtered.done(function( value ) {
 alert( "Value is ( 2*5 = ) 10: " + value );
});
```

Example
Filter reject value:

```
var defer = $.Deferred(),
  filtered = defer.pipe( null, function( value ) {
   return value * 3;
  });

defer.reject( 6 );
filtered.fail(function( value ) {
 alert( "Value is ( 3*6 = ) 18: " + value );
});
```

Example
Chain tasks:

```
var request = $.ajax( url, { dataType: "json" } ),
  chained = request.pipe(function( data ) {
   return $.ajax( url2, { data: { user: data.userId } } );
  });

chained.done(function( data ) {
 // data retrieved from url2 as provided by the first request
});
```

deferred.always(alwaysCallbacks, [alwaysCallbacks])
Add handlers to be called when the Deferred object is either resolved or rejected.

Arguments
alwaysCallbacks - A function, or array of functions, that is called when the Deferred is resolved or rejected.
alwaysCallbacks - Optional additional functions, or arrays of functions, that are called when the Deferred is resolved or rejected.

The argument can be either a single function or an array of functions. When the Deferred is resolved or rejected, the `alwaysCallbacks` are called. Since `deferred.always()` returns the Deferred object, other methods of the Deferred object can be chained to this one, including additional `.always()` methods. When the Deferred is resolved or rejected, callbacks are executed in the order they were added, using the arguments provided to the resolve, reject, resolveWith or rejectWith method calls. For more information, see the documentation for Deferred object.

Example

Since the jQuery.get() method returns a `jqXHR` object, which is derived from a Deferred object, we can attach a callback for both success and error using the `deferred.always()` method.

```
$.get("test.php").always( function() {
 alert("$.get completed with success or error callback arguments");
} );
```

promise([type], [target])

Return a Promise object to observe when all actions of a certain type bound to the collection, queued or not, have finished.

Arguments

type - The type of queue that needs to be observed.
target - Object onto which the promise methods have to be attached

The `.promise()` method returns a dynamically generated Promise that is resolved once all actions of a certain type bound to the collection, queued or not, have ended.

By default, `type` is `"fx"`, which means the returned Promise is resolved when all animations of the selected elements have completed.

Resolve context and sole argument is the collection onto which `.promise()` has been called.

If `target` is provided, `.promise()` will attach the methods onto it and then return this object rather than create a new one. This can be useful to attach the Promise behavior to an object that already exists.

Note: The returned Promise is linked to a Deferred object stored on the `.data()` for an element. Since the `.remove()` method removes the element's data as well as the element itself, it will prevent any of the element's unresolved Promises from resolving. If it is necessary to remove an element from the DOM before its Promise is resolved, use `.detach()` instead and follow with `.removeData()` after resolution.

Example

Using `.promise()` on a collection with no active animation returns a resolved Promise:

```
var div = $( "<div />" );

div.promise().done(function( arg1 ) {
 // will fire right away and alert "true"
 alert( this === div && arg1 === div );
});
```

Example

Resolve the returned Promise when all animations have ended (including those initiated in the animation callback or added later on):

```
$("button").bind( "click", function() {
 $("p").append( "Started...");

 $("div").each(function( i ) {
  $( this ).fadeIn().fadeOut( 1000 * (i+1) );
 });

 $( "div" ).promise().done(function() {
  $( "p" ).append( " Finished! " );
 });
});
```

Example

Resolve the returned Promise using a `$.when()` statement (the `.promise()` method makes it possible to do this with jQuery collections):

```
var effect = function() {
 return $("div").fadeIn(800).delay(1200).fadeOut();
};

$("button").bind( "click", function() {
 $("p").append( " Started... ");
```

```
 $.when( effect() ).done(function() {
  $("p").append(" Finished! ");
 });
});
```

deferred.promise([target])

Return a Deferred's Promise object.

Arguments

target - Object onto which the promise methods have to be attached

The `deferred.promise()` method allows an asynchronous function to prevent other code from interfering with the progress or status of its internal request. The Promise exposes only the Deferred methods needed to attach additional handlers or determine the state (`then`, `done`, `fail`, `always`, `pipe`, `progress`, and `state`), but not ones that change the state (`resolve`, `reject`, `notify`, `resolveWith`, `rejectWith`, and `notifyWith`).

If `target` is provided, `deferred.promise()` will attach the methods onto it and then return this object rather than create a new one. This can be useful to attach the Promise behavior to an object that already exists.

If you are creating a Deferred, keep a reference to the Deferred so that it can be resolved or rejected at some point. Return *only* the Promise object via `deferred.promise()` so other code can register callbacks or inspect the current state.

For more information, see the documentation for Deferred object.

Example

Create a Deferred and set two timer-based functions to either resolve or reject the Deferred after a random interval. Whichever one fires first "wins" and will call one of the callbacks. The second timeout has no effect since the Deferred is already complete (in a resolved or rejected state) from the first timeout action. Also set a timer-based progress notification function, and call a progress handler that adds "working..." to the document body.

```
function asyncEvent(){
  var dfd = new jQuery.Deferred();

  // Resolve after a random interval
  setTimeout(function(){
    dfd.resolve("hurray");
  }, Math.floor(400+Math.random()*2000));

  // Reject after a random interval
  setTimeout(function(){
    dfd.reject("sorry");
  }, Math.floor(400+Math.random()*2000));

  // Show a "working..." message every half-second
  setTimeout(function working(){
    if ( dfd.state() === "pending" ) {
      dfd.notify("working... ");
      setTimeout(working, 500);
    }
  }, 1);

  // Return the Promise so caller can't change the Deferred
  return dfd.promise();
}

// Attach a done, fail, and progress handler for the asyncEvent
$.when( asyncEvent() ).then(
  function(status){
    alert( status+', things are going well' );
  },
  function(status){
    alert( status+', you fail this time' );
  },
```

```
  function(status){
    $("body").append(status);
  }
);
```

Example

Use the target argument to promote an existing object to a Promise:

```
// Existing object
var obj = {
 hello: function( name ) {
  alert( "Hello " + name );
 }
},
// Create a Deferred
defer = $.Deferred();

// Set object as a promise
defer.promise( obj );

// Resolve the deferred
defer.resolve( "John" );

// Use the object as a Promise
obj.done(function( name ) {
 obj.hello( name ); // will alert "Hello John"
}).hello( "Karl" ); // will alert "Hello Karl"
```

jQuery.when(deferreds)

Provides a way to execute callback functions based on one or more objects, usually Deferred objects that represent asynchronous events.

Arguments

deferreds - One or more Deferred objects, or plain JavaScript objects.

If a single Deferred is passed to `jQuery.when`, its Promise object (a subset of the Deferred methods) is returned by the method. Additional methods of the Promise object can be called to attach callbacks, such as deferred.then. When the Deferred is resolved or rejected, usually by the code that created the Deferred originally, the appropriate callbacks will be called. For example, the jqXHR object returned by `jQuery.ajax` is a Deferred and can be used this way:

```
$.when( $.ajax("test.aspx") ).then(function(ajaxArgs){
   alert(ajaxArgs[1]); /* ajaxArgs is [ "success", statusText, jqXHR ] */
});
```

If a single argument is passed to `jQuery.when` and it is not a Deferred, it will be treated as a resolved Deferred and any doneCallbacks attached will be executed immediately. The doneCallbacks are passed the original argument. In this case any failCallbacks you might set are never called since the Deferred is never rejected. For example:

```
$.when( { testing: 123 } ).done(
  function(x){ alert(x.testing); } /* alerts "123" */
);
```

In the case where multiple Deferred objects are passed to `jQuery.when`, the method returns the Promise from a new "master" Deferred object that tracks the aggregate state of all the Deferreds it has been passed. The method will resolve its master Deferred as soon as all the Deferreds resolve, or reject the master Deferred as soon as one of the Deferreds is rejected. If the master Deferred is resolved, it is passed the resolved values of all the Deferreds that were passed to `jQuery.when`. For example, when the Deferreds are `jQuery.ajax()` requests, the arguments will be the jqXHR objects for the requests, in the order they were given in the argument list.

In the multiple-Deferreds case where one of the Deferreds is rejected, `jQuery.when` immediately fires the failCallbacks for its master Deferred. Note that some of the Deferreds may still be unresolved at that point. If you need to perform additional processing for this case, such as canceling any unfinished ajax requests, you can keep references to the underlying jqXHR objects in a closure and inspect/cancel them in the failCallback.

Example

Execute a function after two ajax requests are successful. (See the jQuery.ajax() documentation for a complete description of success and error cases for an ajax request).

```
$.when($.ajax("/page1.php"), $.ajax("/page2.php")).done(function(a1, a2){
  /* a1 and a2 are arguments resolved for the
    page1 and page2 ajax requests, respectively */
  var jqXHR = a1[2]; /* arguments are [ "success", statusText, jqXHR ] */
  if ( /Whip It/.test(jqXHR.responseText) ) {
   alert("First page has 'Whip It' somewhere.");
  }
});
```

Example
Execute the function `myFunc` when both ajax requests are successful, or `myFailure` if either one has an error.

```
$.when($.ajax("/page1.php"), $.ajax("/page2.php"))
 .then(myFunc, myFailure);
```

deferred.resolveWith(context, [args])
Resolve a Deferred object and call any doneCallbacks with the given `context` and `args`.

Arguments
context - Context passed to the doneCallbacks as the `this` object.
args - An optional array of arguments that are passed to the doneCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through deferred.promise().

When the Deferred is resolved, any doneCallbacks added by deferred.then or deferred.done are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.resolve()`. Any doneCallbacks added after the Deferred enters the resolved state are executed immediately when they are added, using the arguments that were passed to the `.resolve()` call. For more information, see the documentation for Deferred object.

deferred.rejectWith(context, [args])
Reject a Deferred object and call any failCallbacks with the given `context` and `args`.

Arguments
context - Context passed to the failCallbacks as the `this` object.
args - An optional array of arguments that are passed to the failCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through deferred.promise().

When the Deferred is rejected, any failCallbacks added by deferred.then or deferred.fail are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `deferred.reject()` call. Any failCallbacks added after the Deferred enters the rejected state are executed immediately when they are added, using the arguments that were passed to the `.reject()` call. For more information, see the documentation for Deferred object.

deferred.fail(failCallbacks, [failCallbacks])
Add handlers to be called when the Deferred object is rejected.

Arguments
failCallbacks - A function, or array of functions, that are called when the Deferred is rejected.
failCallbacks - Optional additional functions, or arrays of functions, that are called when the Deferred is rejected.

The `deferred.fail()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is rejected, the failCallbacks are called. Callbacks are executed in the order they were added. Since `deferred.fail()` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `deferred.fail()` methods. The failCallbacks are executed using the arguments provided to the deferred.reject() or deferred.rejectWith() method call in the order they were added. For more information, see the documentation for Deferred object.

Example
Since the jQuery.get method returns a jqXHR object, which is derived from a Deferred, you can attach a success and failure callback using the `deferred.done()` and `deferred.fail()` methods.

```
$.get("test.php")
 .done(function(){ alert("$.get succeeded"); })
 .fail(function(){ alert("$.get failed!"); });
```

deferred.done(doneCallbacks, [doneCallbacks])
Add handlers to be called when the Deferred object is resolved.

Arguments
doneCallbacks - A function, or array of functions, that are called when the Deferred is resolved.
doneCallbacks - Optional additional functions, or arrays of functions, that are called when the Deferred is resolved.

The `deferred.done()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is resolved, the doneCallbacks are called. Callbacks are executed in the order they were added. Since `deferred.done()` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `.done()` methods. When the Deferred is resolved, doneCallbacks are executed using the arguments provided to the resolve or resolveWith method call in the order they were added. For more information, see the documentation for Deferred object.

Example
Since the jQuery.get method returns a jqXHR object, which is derived from a Deferred object, we can attach a success callback using the `.done()` method.

```
$.get("test.php").done(function() {
 alert("$.get succeeded");
});
```

Example
Resolve a Deferred object when the user clicks a button, triggering a number of callback functions:

```
// 3 functions to call when the Deferred object is resolved
function fn1() {
 $("p").append(" 1 ");
}
function fn2() {
 $("p").append(" 2 ");
}
function fn3(n) {
 $("p").append(n + " 3 " + n);
}

// create a deferred object
var dfd = $.Deferred();

// add handlers to be called when dfd is resolved
dfd
// .done() can take any number of functions or arrays of functions
.done( [fn1, fn2], fn3, [fn2, fn1] )
// we can chain done methods, too
.done(function(n) {
 $("p").append(n + " we're done.");
});

// resolve the Deferred object when the button is clicked
$("button").bind("click", function() {
 dfd.resolve("and");
});
```

deferred.then(doneCallbacks, failCallbacks)

Add handlers to be called when the Deferred object is resolved or rejected.

Arguments
doneCallbacks - A function, or array of functions, called when the Deferred is resolved.
failCallbacks - A function, or array of functions, called when the Deferred is rejected.

All three arguments (including progressCallbacks, as of jQuery 1.7) can be either a single function or an array of functions. The arguments can also be `null` if no callback of that type is desired. Alternatively, use `.done()`, `.fail()` or `.progress()` to set only one type of callback.

When the Deferred is resolved, the doneCallbacks are called. If the Deferred is instead rejected, the failCallbacks are called. As of jQuery 1.7, the `deferred.notify()` or `deferred.notifyWith()` methods can be called to invoke the progressCallbacks as many times as desired before the Deferred is resolved or rejected.

Callbacks are executed in the order they were added. Since `deferred.then` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `.then()` methods. For more information, see the documentation for Deferred object.

Example
Since the jQuery.get method returns a jqXHR object, which is derived from a Deferred object, we can attach handlers using the `.then` method.

```
$.get("test.php").then(
  function(){ alert("$.get succeeded"); },
  function(){ alert("$.get failed!"); }
);
```

deferred.reject(args)
Reject a Deferred object and call any failCallbacks with the given `args`.

Arguments
args - Optional arguments that are passed to the failCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through deferred.promise().

When the Deferred is rejected, any failCallbacks added by deferred.then or deferred.fail are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `deferred.reject()` call. Any failCallbacks added after the Deferred enters the rejected state are executed immediately when they are added, using the arguments that were passed to the `.reject()` call. For more information, see the documentation for Deferred object.

deferred.isRejected()
Determine whether a Deferred object has been rejected.

As of jQuery 1.7 this API has been deprecated; please use deferred.state() instead.

Returns `true` if the Deferred object is in the rejected state, meaning that either deferred.reject() or deferred.rejectWith() has been called for the object and the failCallbacks have been called (or are in the process of being called).

Note that a Deferred object can be in one of three states: pending, resolved, or rejected; use deferred.isResolved() to determine whether the Deferred object is in the resolved state. These methods are primarily useful for debugging, for example to determine whether a Deferred has already been resolved even though you are inside code that intended to reject it.

deferred.isResolved()
Determine whether a Deferred object has been resolved.

As of jQuery 1.7 this API has been deprecated; please use deferred.state() instead.

Returns `true` if the Deferred object is in the resolved state, meaning that either deferred.resolve() or deferred.resolveWith() has been called for the object and the doneCallbacks have been called (or are in the process of being called).

Note that a Deferred object can be in one of three states: pending, resolved, or rejected; use deferred.isRejected() to determine whether the Deferred object is in the rejected state. These methods are primarily useful for debugging, for example to determine whether a Deferred has already been resolved even though you are inside code that intended to reject it.

deferred.resolve(args)

Resolve a Deferred object and call any doneCallbacks with the given `args`.

Arguments

args - Optional arguments that are passed to the doneCallbacks.

When the Deferred is resolved, any doneCallbacks added by deferred.then or deferred.done are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.resolve()`. Any doneCallbacks added after the Deferred enters the resolved state are executed immediately when they are added, using the arguments that were passed to the `.resolve()` call. For more information, see the documentation for Deferred object.

Dimensions

outerWidth([includeMargin])

Get the current computed width for the first element in the set of matched elements, including padding and border.

Arguments

includeMargin - A Boolean indicating whether to include the element's margin in the calculation.

Returns the width of the element, along with left and right padding, border, and optionally margin, in pixels.

If `includeMargin` is omitted or `false`, the padding and border are included in the calculation; if `true`, the margin is also included.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.

Example

Get the outerWidth of a paragraph.

```
var p = $("p:first");
$("p:last").text( "outerWidth:" + p.outerWidth()+ " , outerWidth(true):" + p.outerWidth(true) );
```

outerHeight([includeMargin])

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

Arguments

includeMargin - A Boolean indicating whether to include the element's margin in the calculation.

The top and bottom padding and border are always included in the `.outerHeight()` calculation; if the `includeMargin` argument is set to `true`, the margin (top and bottom) is also included.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.

Example

Get the outerHeight of a paragraph.

```
var p = $("p:first");
$("p:last").text( "outerHeight:" + p.outerHeight() + " , outerHeight(true):" + p.outerHeight(true) );
```

innerWidth()

Get the current computed width for the first element in the set of matched elements, including padding but not border.

This method returns the width of the element, including left and right padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.

Example

Get the innerWidth of a paragraph.

```
var p = $("p:first");
$("p:last").text( "innerWidth:" + p.innerWidth() );
```

innerHeight()

Get the current computed height for the first element in the set of matched elements, including padding but not border.

This method returns the height of the element, including top and bottom padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.

Example

Get the innerHeight of a paragraph.

```
var p = $("p:first");
$("p:last").text( "innerHeight:" + p.innerHeight() );
```

width()

Get the current computed width for the first element in the set of matched elements.

The difference between `.css(width)` and `.width()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.width()` method is recommended when an element's width needs to be used in a mathematical calculation.

This method is also able to find the width of the window and document.

```
$(window).width();  // returns width of browser viewport
$(document).width(); // returns width of HTML document
```

Note that `.width()` will always return the content width, regardless of the value of the CSS `box-sizing` property.

Example

Show various widths. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showWidth(ele, w) {
 $("div").text("The width for the " + ele +
        " is " + w + "px.");
}
$("#getp").click(function () {
 showWidth("paragraph", $("p").width());
});
$("#getd").click(function () {
 showWidth("document", $(document).width());
});
$("#getw").click(function () {
 showWidth("window", $(window).width());
});
```

width(value)

Set the CSS width of each element in the set of matched elements.

Arguments

value - An integer representing the number of pixels, or an integer along with an optional unit of measure appended (as a string).

When calling `.width("value")`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, any valid CSS measurement may be used for the width (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS width property does not include padding, border, or margin, unless the `box-sizing` CSS property is used.

If no explicit unit is specified (like "em" or "%") then "px" is assumed.

Note that `.width("value")` sets the width of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the `outerWidth` of the box instead of the content width.

Example
Change the width of each div the first time it is clicked (and change its color).

```
(function() {
 var modWidth = 50;
 $("div").one('click', function () {
  $(this).width(modWidth).addClass("mod");
 modWidth -= 8;
 });
})();
```

height()
Get the current computed height for the first element in the set of matched elements.

The difference between `.css('height')` and `.height()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.height()` method is recommended when an element's height needs to be used in a mathematical calculation.

This method is also able to find the height of the window and document.

```
$(window).height();  // returns height of browser viewport
$(document).height(); // returns height of HTML document
```

Note that `.height()` will always return the content height, regardless of the value of the CSS `box-sizing` property.

Note: Although `style` and `script` tags will report a value for `.width()` or `height()` when absolutely positioned and given `display:block`, it is strongly discouraged to call those methods on these tags. In addition to being a bad practice, the results may also prove unreliable.

Example
Show various heights. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showHeight(ele, h) {
   $("div").text("The height for the " + ele +
          " is " + h + "px.");
  }
  $("#getp").click(function () {
   showHeight("paragraph", $("p").height());
  });
  $("#getd").click(function () {
   showHeight("document", $(document).height());
  });
  $("#getw").click(function () {
   showHeight("window", $(window).height());
  });
```

height(value)
Set the CSS height of every matched element.

Arguments
value - An integer representing the number of pixels, or an integer with an optional unit of measure appended (as a string).

When calling `.height(value)`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, a valid CSS measurement must be provided for the height (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS height property does not include padding, border, or margin.

If no explicit unit was specified (like 'em' or '%') then "px" is concatenated to the value.

Note that `.height(value)` sets the height of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the outerHeight of the box instead of the content height.

Example

To set the height of each div on click to 30px plus a color change.

```
$("div").one('click', function () {
   $(this).height(30)
       .css({cursor:"auto", backgroundColor:"green"});
  });
```

Effects

fadeToggle([duration], [easing], [callback])
Display or hide the matched elements by animating their opacity.

Arguments
duration - A string or number determining how long the animation will run.
easing - A string indicating which easing function to use for the transition.
callback - A function to call once the animation is complete.

The `.fadeToggle()` method animates the opacity of the matched elements. When called on a visible element, the element's `display` style property is set to `none` once the opacity reaches 0, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Easing

The string representing an easing function specifies the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example
Fades first paragraph in or out, completing the animation within 600 milliseconds and using a linear easing. Fades last paragraph in or out for 200 milliseconds, inserting a "finished" message upon completion.

```
$("button:first").click(function() {
 $("p:first").fadeToggle("slow", "linear");
});
$("button:last").click(function () {
 $("p:last").fadeToggle("fast", function () {
  $("#log").append("<div>finished</div>");
 });
});
```

Basics

toggle([duration], [callback])
Display or hide the matched elements.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

Note: The event handling suite also has a method named .toggle(). Which one is fired depends on the set of arguments passed.

With no parameters, the `.toggle()` method simply toggles the visibility of elements:

```
$('.target').toggle();
```

The matched elements will be revealed or hidden immediately, with no animation, by changing the CSS `display` property. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration is provided, `.toggle()` becomes an animation method. The `.toggle()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

We will cause `.toggle()` to be called when another element is clicked:

```
$('#clickme').click(function() {
 $('#book').toggle('slow', function() {
  // Animation complete.
 });
});
```

With the element initially shown, we can hide it slowly with the first click:

A second click will show the element once again:

The second version of the method accepts a Boolean parameter. If this parameter is `true`, then the matched elements are shown; if `false`, the elements are hidden. In essence, the statement:

```
$('#foo').toggle(showOrHide);
```

is equivalent to:

```
if ( showOrHide == true ) {
 $('#foo').show();
} else if ( showOrHide == false ) {
 $('#foo').hide();
}
```

Example

Toggles all paragraphs.

```
$("button").click(function () {
$("p").toggle();
```

```
});
```

Example
Animates all paragraphs to be shown if they are hidden and hidden if they are visible, completing the animation within 600 milliseconds.

```
$("button").click(function () {
$("p").toggle("slow");
});
```

Example
Shows all paragraphs, then hides them all, back and forth.

```
var flip = 0;
$("button").click(function () {
$("p").toggle( flip++ % 2 == 0 );
});
```

hide()
Hide the matched elements.

With no parameters, the `.hide()` method is the simplest way to hide an element:

```
$('.target').hide();
```

The matched elements will be hidden immediately, with no animation. This is roughly equivalent to calling `.css('display', 'none')`, except that the value of the `display` property is saved in jQuery's data cache so that `display` can later be restored to its initial value. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration is provided, `.hide()` becomes an animation method. The `.hide()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Note that `.hide()` is fired immediately and will override the animation queue if no duration or a duration of 0 is specified.

As of jQuery **1.4.3**, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
With the element initially shown, we can hide it slowly:
$('#clickme').click(function() {
 $('#book').hide('slow', function() {
  alert('Animation complete.');
 });
});
```

Example
Hides all paragraphs then the link on click.

```
$("p").hide();
  $("a").click(function ( event ) {
   event.preventDefault();
   $(this).hide();
  });
```

Example
Animates all shown paragraphs to hide slowly, completing the animation within 600 milliseconds.

```
$("button").click(function () {
   $("p").hide("slow");
  });
```

Example
Animates all spans (words in this case) to hide fastly, completing each animation within 200 milliseconds. Once each animation is done, it starts the next one.

```
$("#hidr").click(function () {
   $("span:last-child").hide("fast", function () {
    // use callee so don't have to name the function
    $(this).prev().hide("fast", arguments.callee);
   });
  });
  $("#showr").click(function () {
   $("span").show(2000);
  });
```

Example
Hides the divs when clicked over 2 seconds, then removes the div element when its hidden. Try clicking on more than one box at a time.

```
for (var i = 0; i < 5; i++) {
   $("<div>").appendTo(document.body);
  }
  $("div").click(function () {
   $(this).hide(2000, function () {
    $(this).remove();
   });
  });
```

show()
Display the matched elements.

With no parameters, the `.show()` method is the simplest way to display an element:

```
$('.target').show();
```

The matched elements will be revealed immediately, with no animation. This is roughly equivalent to calling `.css('display', 'block')`, except that the `display` property is restored to whatever it was initially. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

Note: If using !important in your styles, such as`display: none !important`,it is necessary to override the style using `.css('display', 'block !important')` should you wish for `.show()` to function correctly.

When a duration is provided, `.show()` becomes an animation method. The `.show()` method animates the width, height, and opacity of the matched elements simultaneously.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to

indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
With the element initially hidden, we can show it slowly:
$('#clickme').click(function() {
 $('#book').show('slow', function() {
  // Animation complete.
 });
});
```

Example
Animates all hidden paragraphs to show slowly, completing the animation within 600 milliseconds.

```
$("button").click(function () {
  $("p").show("slow");
  });
```

Example
Show the first div, followed by each next adjacent sibling div in order, with a 200ms animation. Each animation starts when the previous sibling div's animation ends.

```
$("#showr").click(function () {
 $("div").first().show("fast", function showNext() {
  $(this).next("div").show("fast", showNext);
 });
});

$("#hidr").click(function () {
 $("div").hide(1000);
});
```

Example
Show all span and input elements with an animation. Change the text once the animation is done.

```
function doIt() {
 $("span,div").show("slow");
}
/* can pass in function name */
$("button").click(doIt);

$("form").submit(function () {
 if ($("input").val() == "yes") {
  $("p").show(4000, function () {
   $(this).text("Ok, DONE! (now showing)");
  });
 }
 $("span,div").hide("fast");
 /* to stop the submit */
```

```
 return false;
});
```

Custom

jQuery.fx.interval
The rate (in milliseconds) at which animations fire.

This property can be manipulated to adjust the number of frames per second at which animations will run. The default is 13 milliseconds. Making this a lower number could make the animations run smoother in faster browsers (such as Chrome) but there may be performance and CPU implications of doing so.

Since jQuery uses one global interval, no animation should be running or all animations should stop for the change of this property to take effect.

Note: `jQuery.fx.interval` currently has no effect in browsers that support the `requestAnimationFrame` property, such as Google Chrome 11. This behavior is subject to change in a future release.

Example
Cause all animations to run with less frames.

```
jQuery.fx.interval = 100;

$("input").click(function(){
 $("div").toggle( 3000 );
});
```

delay(duration, [queueName])
Set a timer to delay execution of subsequent items in the queue.

Arguments
duration - An integer indicating the number of milliseconds to delay execution of the next item in the queue.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Added to jQuery in version 1.4, the `.delay()` method allows us to delay the execution of functions that follow it in the queue. It can be used with the standard effects queue or with a custom queue. Only subsequent events in a queue are delayed; for example this will *not* delay the no-arguments forms of `.show()` or `.hide()` which do not use the effects queue.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of 200 and 600 milliseconds, respectively.

Using the standard effects queue, we can, for example, set an 800-millisecond delay between the `.slideUp()` and `.fadeIn()` of `<div id="foo">`:

```
$('#foo').slideUp(300).delay(800).fadeIn(400);
```

When this statement is executed, the element slides up for 300 milliseconds and then pauses for 800 milliseconds before fading in for 400 milliseconds.

The `.delay()` method is best for delaying between queued jQuery effects. Because it is limited-it doesn't, for example, offer a way to cancel the delay-`.delay()` is not a replacement for JavaScript's native setTimeout function, which may be more appropriate for certain use cases.

Example
Animate the hiding and showing of two divs, delaying the first before showing it.

```
$("button").click(function() {
   $("div.first").slideUp(300).delay(800).fadeIn(400);
   $("div.second").slideUp(300).fadeIn(400);
  });
```

jQuery.fx.off
Globally disable all animations.

When this property is set to `true`, all animation methods will immediately set elements to their final state when called, rather than displaying an effect. This may be desirable for a couple reasons:

- jQuery is being used on a low-resource device.
- Users are encountering accessibility problems with the animations (see the article Turn Off Animation for more information).

Animations can be turned back on by setting the property to `false`.

Example
Toggle animation on and off

```
var toggleFx = function() {
 $.fx.off = !$.fx.off;
};
toggleFx();

$("button").click(toggleFx)

$("input").click(function(){
 $("div").toggle("slow");
});
```

clearQueue([queueName])
Remove from the queue all items that have not yet been run.

Arguments
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

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.

Example
Empty the queue.

```
$("#start").click(function () {

 var myDiv = $("div");
 myDiv.show("slow");
 myDiv.animate({left:'+=200'},5000);
 myDiv.queue(function () {
  var _this = $(this);
  _this.addClass("newcolor");
  _this.dequeue();
 });

 myDiv.animate({left:'-=200'},1500);
 myDiv.queue(function () {
  var _this = $(this);
  _this.removeClass("newcolor");
  _this.dequeue();
 });
 myDiv.slideUp();

});

$("#stop").click(function () {
 var myDiv = $("div");
```

```
 myDiv.clearQueue();
 myDiv.stop();
});
```

dequeue([queueName])

Execute the next function on the queue for the matched elements.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

When `.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 `.dequeue()` to be called, so that the sequence can continue.

Example

Use dequeue to end a custom queue function which allows the queue to keep going.

```
$("button").click(function () {
 $("div").animate({left:'+=200px'}, 2000);
 $("div").animate({top:'0px'}, 600);
 $("div").queue(function () {
  $(this).toggleClass("red");
  $(this).dequeue();
 });
 $("div").animate({left:'10px', top:'30px'}, 700);
});
```

queue([queueName])

Show the queue of functions to be executed on the matched elements.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Example

Show the length of the queue.

```
var div = $("div");

function runIt() {
 div.show("slow");
 div.animate({left:'+=200'},2000);
 div.slideToggle(1000);
 div.slideToggle("fast");
 div.animate({left:'-=200'},1500);
 div.hide("slow");
 div.show(1200);
 div.slideUp("normal", runIt);
}

function showIt() {
 var n = div.queue("fx");
 $("span").text( n.length );
 setTimeout(showIt, 100);
}

runIt();
showIt();
```

queue([queueName], newQueue)

Manipulate the queue of functions to be executed on the matched elements.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.
newQueue - An array of functions to replace the current queue contents.

Every element can have one to many queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution. The typical example of this is calling multiple animation methods on an element. For example:

```
$('#foo').slideUp().fadeIn();
```

When this statement is executed, the element begins its sliding animation immediately, but the fading transition is placed on the `fx` queue to be called only once the sliding transition is complete.

The `.queue()` method allows us to directly manipulate this queue of functions. Calling `.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue.

This feature is similar to providing a callback function with an animation method, but does not require the callback to be given at the time the animation is performed.

```
$('#foo').slideUp();
$('#foo').queue(function() {
 alert('Animation complete.');
 $(this).dequeue();
});
```

This is equivalent to:

```
$('#foo').slideUp(function() {
 alert('Animation complete.');
});
```

Note that when adding a function with `.queue()`, we should ensure that `.dequeue()` is eventually called so that the next function in line executes.

As of jQuery 1.4, the function that's called is passed another function as the first argument. When called, this automatically dequeues the next item and keeps the queue moving. We use it as follows:

```
$("#test").queue(function(next) {
  // Do some stuff...
  next();
});
```

Example

Queue a custom function.

```
$(document.body).click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
   });
   $("div").animate({left:'-=200'},500);
   $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
   });
   $("div").slideUp();
  });
```

Example

Set a queue array to delete the queue.

```
$("#start").click(function () {
```

```
  $("div").show("slow");
  $("div").animate({left:'+=200'},5000);
  $("div").queue(function () {
   $(this).addClass("newcolor");
   $(this).dequeue();
  });
  $("div").animate({left:'-=200'},1500);
  $("div").queue(function () {
   $(this).removeClass("newcolor");
   $(this).dequeue();
  });
  $("div").slideUp();
 });
 $("#stop").click(function () {
  $("div").queue("fx", []);
  $("div").stop();
 });
```

stop([clearQueue], [jumpToEnd])

Stop the currently-running animation on the matched elements.

Arguments

clearQueue - A Boolean indicating whether to remove queued animation as well. Defaults to `false`.
jumpToEnd - A Boolean indicating whether to complete the current animation immediately. Defaults to `false`.

When `.stop()` is called on an element, the currently-running animation (if any) is immediately stopped. If, for instance, an element is being hidden with `.slideUp()` when `.stop()` is called, the element will now still be displayed, but will be a fraction of its previous height. Callback functions are not called.

If more than one animation method is called on the same element, the later animations are placed in the effects queue for the element. These animations will not begin until the first one completes. When `.stop()` is called, the next animation in the queue begins immediately. If the `clearQueue` parameter is provided with a value of `true`, then the rest of the animations in the queue are removed and never run.

If the `jumpToEnd` argument is provided with a value of `true`, the current animation stops, but the element is immediately given its target values for each CSS property. In our above `.slideUp()` example, the element would be immediately hidden. The callback function is then immediately called, if provided.

As of jQuery 1.7, if the first argument is provided as a string, only the animations in the queue represented by that string will be stopped.

The usefulness of the `.stop()` method is evident when we need to animate an element on `mouseenter` and `mouseleave`:

```
<div id="hoverme">
 Hover me
 <img id="hoverme" src="book.png" alt="" width="100" height="123" />
</div>
```

We can create a nice fade effect without the common problem of multiple queued animations by adding `.stop(true, true)` to the chain:

```
$('#hoverme-stop-2').hover(function() {
 $(this).find('img').stop(true, true).fadeOut();
}, function() {
 $(this).find('img').stop(true, true).fadeIn();
});
```

Toggling Animations

As of jQuery 1.7, stopping a toggled animation prematurely with `.stop()` will trigger jQuery's internal effects tracking. In previous versions, calling the `.stop()` method before a toggled animation was completed would cause the animation to lose track of its state (if jumpToEnd was false). Any subsequent animations would start at a new "half-way" state, sometimes resulting in the element disappearing. To observe the new behavior, see the final example below.

Animations may be stopped globally by setting the property `$.fx.off` to `true`. When this is done, all animation methods will immediately set elements to their final state when called, rather than displaying an effect.

Example
Click the Go button once to start the animation, then click the STOP button to stop it where it's currently positioned. Another option is to click several buttons to queue them up and see that stop just kills the currently playing one.

```
/* Start animation */
$("#go").click(function(){
$(".block").animate({left: '+=100px'}, 2000);
});

/* Stop animation when button is clicked */
$("#stop").click(function(){
$(".block").stop();
});

/* Start animation in the opposite direction */
$("#back").click(function(){
$(".block").animate({left: '-=100px'}, 2000);
});
```

Example
Click the slideToggle button to start the animation, then click again before the animation is completed. The animation will toggle the other direction from the saved starting point.

```
var $block = $('.block');
/* Toggle a sliding animation animation */
$('#toggle').on('click', function() {
  $block.stop().slideToggle(1000);
});
```

animate(properties, [duration], [easing], [complete])
Perform a custom animation of a set of CSS properties.

Arguments
properties - A map of CSS properties that the animation will move toward.
duration - A string or number determining how long the animation will run.
easing - A string indicating which easing function to use for the transition.
complete - A function to call once the animation is complete.

The `.animate()` method allows us to create animation effects on any numeric CSS property. The only required parameter is a map of CSS properties. This map is similar to the one that can be sent to the `.css()` method, except that the range of properties is more restrictive.

Animation Properties and Values

All animated properties should be animated to a *single numeric value*, except as noted below; most properties that are non-numeric cannot be animated using basic jQuery functionality (For example, `width`, `height`, or `left` can be animated but `background-color` cannot be, unless the jQuery.Color() plugin is used). Property values are treated as a number of pixels unless otherwise specified. The units `em` and `%` can be specified where applicable.

In addition to style properties, some non-style properties such as `scrollTop` and `scrollLeft`, as well as custom properties, can be animated.

Shorthand CSS properties (e.g. font, background, border) are not fully supported. For example, if you want to animate the rendered border width, at least a border style and border width other than "auto" must be set in advance. Or, if you want to animate font size, you would use `fontSize` or the CSS equivalent `'font-size'` rather than simply `'font'`.

In addition to numeric values, each property can take the strings `'show'`, `'hide'`, and `'toggle'`. These shortcuts allow for custom hiding and showing animations that take into account the display type of the element.

Animated properties can also be relative. If a value is supplied with a leading += or -= sequence of characters, then the target value is computed by

adding or subtracting the given number from the current value of the property.

Note: Unlike shorthand animation methods such as `.slideDown()` and `.fadeIn()`, the `.animate()` method does *not* make hidden elements visible as part of the effect. For example, given `$('someElement').hide().animate({height:'20px'}, 500)`, the animation will run, but *the element will remain hidden*.

Duration

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The default duration is `400` milliseconds. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Complete Function

If supplied, the `complete` callback function is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, the callback is executed once per matched element, not once for the animation as a whole.

Basic Usage

To animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123"
 style="position: relative; left: 10px;" />
```

To animate the opacity, left offset, and height of the image simultaneously:

```
$('#clickme').click(function() {
 $('#book').animate({
  opacity: 0.25,
  left: '+=50',
  height: 'toggle'
 }, 5000, function() {
  // Animation complete.
 });
});
```

Note that the target value of the `height` property is `'toggle'`. Since the image was visible before, the animation shrinks the height to 0 to hide it. A second click then reverses this transition:

The `opacity` of the image is already at its target value, so this property is not animated by the second click. Since the target value for `left` is a relative value, the image moves even farther to the right during this second animation.

Directional properties (`top`, `right`, `bottom`, `left`) have no discernible effect on elements if their `position` style property is `static`, which it is by default.

Note: The jQuery UI project extends the `.animate()` method by allowing some non-numeric styles such as colors to be animated. The project also includes mechanisms for specifying animations through CSS classes rather than individual attributes.

Note: if attempting to animate an element with a height or width of 0px, where contents of the element are visible due to overflow, jQuery may clip this overflow during animation. By fixing the dimensions of the original element being hidden however, it is possible to ensure that the animation runs smoothly. A clearfix can be used to automatically fix the dimensions of your main element without the need to set this manually.

Step Function

The second version of `.animate()` provides a `step` option - a callback function that is fired at each step of the animation. This function is useful for enabling custom animation types or altering the animation as it is occurring. It accepts two arguments (`now` and `fx`), and `this` is set to the DOM element being animated.

- `now`: the numeric value of the property being animated at each step
- `fx`: a reference to the `jQuery.fx` prototype object, which contains a number of properties such as `elem` for the animated element, `start` and `end` for the first and last value of the animated property, respectively, and `prop` for the property being animated.

Note that the `step` function is called for each animated property on each animated element. For example, given two list items, the `step` function fires four times at each step of the animation:

```
$('li').animate({
 opacity: .5,
 height: '50%'
},
{
 step: function(now, fx) {
  var data = fx.elem.id + ' ' + fx.prop + ': ' + now;
  $('body').append('<div>' + data + '</div>');
 }
});
```

Easing

The remaining parameter of `.animate()` is a string naming an easing function to use. An easing function specifies the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Per-property Easing

As of jQuery version 1.4, you can set per-property easing functions within a single `.animate()` call. In the first version of `.animate()`, each property can take an array as its value: The first member of the array is the CSS property and the second member is an easing function. If a per-property easing function is not defined for a particular property, it uses the value of the `.animate()` method's optional easing argument. If the easing argument is not defined, the default `swing` function is used.

For example, to simultaneously animate the width and height with the `swing` easing function and the opacity with the `linear` easing function:

```
$('#clickme').click(function() {
 $('#book').animate({
  width: ['toggle', 'swing'],
  height: ['toggle', 'swing'],
  opacity: 'toggle'
 }, 5000, 'linear', function() {
   $(this).after('<div>Animation complete.</div>');
 });
});
```

In the second version of `.animate()`, the options map can include the `specialEasing` property, which is itself a map of CSS properties and their corresponding easing functions. For example, to simultaneously animate the width using the `linear` easing function and the height using the `easeOutBounce` easing function:

```
$('#clickme').click(function() {
 $('#book').animate({
  width: 'toggle',
  height: 'toggle'
 }, {
  duration: 5000,
  specialEasing: {
   width: 'linear',
```

```
   height: 'easeOutBounce'
  },
  complete: function() {
   $(this).after('<div>Animation complete.</div>');
  }
 });
});
```

As previously noted, a plugin is required for the `easeOutBounce` function.

Example

Click the button to animate the div with a number of different properties.

```
/* Using multiple unit types within one animation. */

$("#go").click(function(){
 $("#block").animate({
  width: "70%",
  opacity: 0.4,
  marginLeft: "0.6in",
  fontSize: "3em",
  borderWidth: "10px"
 }, 1500 );
});
```

Example

Animates a div's left property with a relative value. Click several times on the buttons to see the relative animations queued up.

```
$("#right").click(function(){
 $(".block").animate({"left": "+=50px"}, "slow");
});

$("#left").click(function(){
 $(".block").animate({"left": "-=50px"}, "slow");
});
```

Example

The first button shows how an unqueued animation works. It expands the div out to 90% width **while** the font-size is increasing. Once the font-size change is complete, the border animation will begin.The second button starts a traditional chained animation, where each animation will start once the previous animation on the element has completed.

```
$( "#go1" ).click(function(){
 $( "#block1" ).animate( { width: "90%" }, { queue: false, duration: 3000 })
   .animate({ fontSize: "24px" }, 1500 )
   .animate({ borderRightWidth: "15px" }, 1500 );
});

$( "#go2" ).click(function(){
 $( "#block2" ).animate({ width: "90%" }, 1000 )
   .animate({ fontSize: "24px" }, 1000 )
   .animate({ borderLeftWidth: "15px" }, 1000 );
});

$( "#go3" ).click(function(){
 $( "#go1" ).add( "#go2" ).click();
});

$( "#go4" ).click(function(){
 $( "div" ).css({ width: "", fontSize: "", borderWidth: "" });
});
```

Example

Animates the first div's left property and synchronizes the remaining divs, using the step function to set their left properties at each stage of the

animation.

```
$( "#go" ).click(function(){
 $( ".block:first" ).animate({
  left: 100
 }, {
  duration: 1000,
  step: function( now, fx ){
   $( ".block:gt(0)" ).css( "left", now );
  }
 });
});
```

Example
Animate all paragraphs to toggle both height and opacity, completing the animation within 600 milliseconds.

```
$( "p" ).animate({
 height: "toggle", opacity: "toggle"
}, "slow" );
```

Example
Animate all paragraphs to a left style of 50 and opacity of 1 (opaque, visible), completing the animation within 500 milliseconds.

```
$( "p" ).animate({
 left: 50, opacity: 1
}, 500 );
```

Example
Animate the left and opacity style properties of all paragraphs; run the animation *outside* the queue, so that it will automatically start without waiting for its turn.

```
$( "p" ).animate({
 left: "50px", opacity: 1
}, { duration: 500, queue: false });
```

Example
An example of using an 'easing' function to provide a different style of animation. This will only work if you have a plugin that provides this easing function. Note, this code will do nothing unless the paragraph element is hidden.

```
$( "p" ).animate({
 opacity: "show"
}, "slow", "easein" );
```

Example
Animates all paragraphs to toggle both height and opacity, completing the animation within 600 milliseconds.

```
$( "p" ).animate({
 height: "toggle", opacity: "toggle"
}, { duration: "slow" });
```

Example
Use an easing function to provide a different style of animation. This will only work if you have a plugin that provides this easing function.

```
$( "p" ).animate({
 opacity: "show"
}, { duration: "slow", easing: "easein" });
```

Example
Animate all paragraphs and execute a callback function when the animation is complete. The first argument is a map of CSS properties, the second specifies that the animation should take 1000 milliseconds to complete, the third states the easing type, and the fourth argument is an anonymous callback function.

```
$( "p" ).animate({
```

```
 height: 200, width: 400, opacity: 0.5
}, 1000, "linear", function() {
 alert("all done");
});
```

Fading

fadeToggle([duration], [easing], [callback])

Display or hide the matched elements by animating their opacity.

Arguments

duration - A string or number determining how long the animation will run.
easing - A string indicating which easing function to use for the transition.
callback - A function to call once the animation is complete.

The `.fadeToggle()` method animates the opacity of the matched elements. When called on a visible element, the element's `display` style property is set to `none` once the opacity reaches 0, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Easing

The string representing an easing function specifies the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Fades first paragraph in or out, completing the animation within 600 milliseconds and using a linear easing. Fades last paragraph in or out for 200 milliseconds, inserting a "finished" message upon completion.

```
$("button:first").click(function() {
 $("p:first").fadeToggle("slow", "linear");
});
$("button:last").click(function () {
 $("p:last").fadeToggle("fast", function () {
  $("#log").append("<div>finished</div>");
 });
});
```

fadeTo(duration, opacity, [callback])

Adjust the opacity of the matched elements.

Arguments

duration - A string or number determining how long the animation will run.
opacity - A number between 0 and 1 denoting the target opacity.
callback - A function to call once the animation is complete.

The `.fadeTo()` method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, the default duration of `400` milliseconds is used. Unlike the other effect methods, `.fadeTo()` requires that `duration` be explicitly specified.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
  Click here
 </div>
 <img id="book" src="book.png" alt="" width="100" height="123" />
 With the element initially shown, we can dim it slowly:
 $('#clickme').click(function() {
  $('#book').fadeTo('slow', 0.5, function() {
   // Animation complete.
  });
 });
```

With `duration` set to `0`, this method just changes the `opacity` CSS property, so `.fadeTo(0, opacity)` is the same as `.css('opacity', opacity)`.

Example

Animates first paragraph to fade to an opacity of 0.33 (33%, about one third visible), completing the animation within 600 milliseconds.

```
$("p:first").click(function () {
$(this).fadeTo("slow", 0.33);
});
```

Example

Fade div to a random opacity on each click, completing the animation within 200 milliseconds.

```
$("div").click(function () {
$(this).fadeTo("fast", Math.random());
});
```

Example

Find the right answer! The fade will take 250 milliseconds and change various styles when it completes.

```
var getPos = function (n) {
return (Math.floor(n) * 90) + "px";
};
$("p").each(function (n) {
var r = Math.floor(Math.random() * 3);
var tmp = $(this).text();
$(this).text($("p:eq(" + r + ")").text());
$("p:eq(" + r + ")").text(tmp);
$(this).css("left", getPos(n));
});
$("div").each(function (n) {
   $(this).css("left", getPos(n));
  })
.css("cursor", "pointer")
.click(function () {
   $(this).fadeTo(250, 0.25, function () {
      $(this).css("cursor", "")
          .prev().css({"font-weight": "bolder",
                "font-style": "italic"});
```

```
    });
  });
```

fadeOut([duration], [callback])

Hide the matched elements by fading them to transparent.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.fadeOut()` method animates the opacity of the matched elements. Once the opacity reaches 0, the `display` style property is set to `none`, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially shown, we can hide it slowly:

```
$('#clickme').click(function() {
 $('#book').fadeOut('slow', function() {
  // Animation complete.
 });
});
```

Note: To avoid unnecessary DOM manipulation, `.fadeOut()` will not hide an element that is already considered hidden. For information on which elements jQuery considers hidden, see :hidden Selector.

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all paragraphs to fade out, completing the animation within 600 milliseconds.

```
$("p").click(function () {
 $("p").fadeOut("slow");
 });
```

Example

Fades out spans in one section that you click on.

```
$("span").click(function () {
 $(this).fadeOut(1000, function () {
 $("div").text("'" + $(this).text() + "' has faded!");
 $(this).remove();
 });
 });
 $("span").hover(function () {
 $(this).addClass("hilite");
 }, function () {
 $(this).removeClass("hilite");
 });
```

Example

Fades out two divs, one with a "linear" easing and one with the default, "swing," easing.

```
$("#btn1").click(function() {
 function complete() {
  $("<div/>").text(this.id).appendTo("#log");
 }

 $("#box1").fadeOut(1600, "linear", complete);
 $("#box2").fadeOut(1600, complete);
});

$("#btn2").click(function() {
 $("div").show();
 $("#log").empty();
});
```

fadeIn([duration], [callback])

Display the matched elements by fading them to opaque.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.fadeIn()` method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
   Click here
  </div>
  <img id="book" src="book.png" alt="" width="100" height="123" />
  With the element initially hidden, we can show it slowly:
  $('#clickme').click(function() {
   $('#book').fadeIn('slow', function() {
    // Animation complete
   });
  });
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example
Animates hidden divs to fade in one by one, completing each animation within 600 milliseconds.

```
$(document.body).click(function () {
    $("div:hidden:first").fadeIn("slow");
   });
```

Example
Fades a red block in over the text. Once the animation is done, it quickly fades in more text on top.

```
$("a").click(function () {
     $("div").fadeIn(3000, function () {
      $("span").fadeIn(100);
     });
     return false;
    });
```

Sliding

slideToggle([duration], [callback])
Display or hide the matched elements with a sliding motion.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideToggle()` method animates the height of the matched elements. This causes lower parts of the page to slide up or down, appearing to reveal or conceal the items. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`. When the height reaches 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

We will cause `.slideToggle()` to be called when another element is clicked:

```
$('#clickme').click(function() {
 $('#book').slideToggle('slow', function() {
  // Animation complete.
```

```
 });
});
```

With the element initially shown, we can hide it slowly with the first click:

A second click will show the element once again:

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all paragraphs to slide up or down, completing the animation within 600 milliseconds.

```
$("button").click(function () {
   $("p").slideToggle("slow");
  });
```

Example

Animates divs between dividers with a toggle that makes some appear and some disappear.

```
$("#aa").click(function () {
  $("div:not(.still)").slideToggle("slow", function () {
   var n = parseInt($("span").text(), 10);
   $("span").text(n + 1);
  });
 });
```

slideUp([duration], [callback])

Hide the matched elements with a sliding motion.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideUp()` method animates the height of the matched elements. This causes lower parts of the page to slide up, appearing to conceal the items. Once the height reaches 0 (or, if set, to whatever the CSS min-height property is), the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially shown, we can hide it slowly:

```
$('#clickme').click(function() {
 $('#book').slideUp('slow', function() {
  // Animation complete.
 });
});
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all divs to slide up, completing the animation within 400 milliseconds.

```
$(document.body).click(function () {
  if ($("div:first").is(":hidden")) {
   $("div").show("slow");
  } else {
   $("div").slideUp();
  }
 });
```

Example

Animates the parent paragraph to slide up, completing the animation within 200 milliseconds. Once the animation is done, it displays an alert.

```
$("button").click(function () {
  $(this).parent().slideUp("slow", function () {
   $("#msg").text($("button", this).text() + " has completed.");
  });
 });
```

slideDown([duration], [callback])

Display the matched elements with a sliding motion.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideDown()` method animates the height of the matched elements. This causes lower parts of the page to slide down, making way for the revealed items.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially hidden, we can show it slowly:

```
$('#clickme').click(function() {
 $('#book').slideDown('slow', function() {
  // Animation complete.
 });
});
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all divs to slide down and show themselves over 600 milliseconds.

```
$(document.body).click(function () {
if ($("div:first").is(":hidden")) {
$("div").slideDown("slow");
} else {
$("div").hide();
}
});
```

Example

Animates all inputs to slide down, completing the animation within 1000 milliseconds. Once the animation is done, the input look is changed especially if it is the middle input which gets the focus.

```
$("div").click(function () {
$(this).css({ borderStyle:"inset", cursor:"wait" });
$("input").slideDown(1000,function(){
$(this).css("border", "2px red inset")
.filter(".middle")
 .css("background", "yellow")
 .focus();
$("div").css("visibility", "hidden");
});
});
```

Events

event.delegateTarget
The element where the currently-called jQuery event handler was attached.

This property is most often useful in delegated events attached by .delegate() or .on(), where the event handler is attached at an ancestor of the element being processed. It can be used, for example, to identify and remove event handlers at the delegation point.

For non-delegated event handlers attached directly to an element, `event.delegateTarget` will always be equal to `event.currentTarget`.

Example
When a button in any box class is clicked, change the box's background color to red.

```
$(".box").on("click", "button", function(event) {
 $(event.delegateTarget).css("background-color", "red");
});
```

Browser Events

scroll(handler(eventObject))
Bind an event handler to the "scroll" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('scroll', handler)` in the first and second variations, and `.trigger('scroll')` in the third.

The `scroll` event is sent to an element when the user scrolls to a different place in the element. It applies to `window` objects, but also to scrollable frames and elements with the `overflow` CSS property set to `scroll` (or `auto` when the element's explicit height or width is less than the height or width of its contents).

For example, consider the HTML:

```
<div id="target" style="overflow: scroll; width: 200px; height: 100px;">
 Lorem ipsum dolor sit amet, consectetur adipisicing elit,
 sed do eiusmod tempor incididunt ut labore et dolore magna
 aliqua. Ut enim ad minim veniam, quis nostrud exercitation
 ullamco laboris nisi ut aliquip ex ea commodo consequat.
 Duis aute irure dolor in reprehenderit in voluptate velit
 esse cillum dolore eu fugiat nulla pariatur. Excepteur
 sint occaecat cupidatat non proident, sunt in culpa qui
 officia deserunt mollit anim id est laborum.
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The style definition is present to make the target element small enough to be scrollable:

The `scroll` event handler can be bound to this element:

```
$('#target').scroll(function() {
 $('#log').append('<div>Handler for .scroll() called.</div>');
});
```

Now when the user scrolls the text up or down, one or more messages are appended to `<div id="log"></div>`:

Handler for .scroll() called.

To trigger the event manually, apply `.scroll()` without an argument:

```
$('#other').click(function() {
 $('#target').scroll();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

A `scroll` event is sent whenever the element's scroll position changes, regardless of the cause. A mouse click or drag on the scroll bar, dragging inside the element, pressing the arrow keys, or using the mouse's scroll wheel could cause this event.

Example
To do something when your page is scrolled:

```
$("p").clone().appendTo(document.body);
  $("p").clone().appendTo(document.body);
  $("p").clone().appendTo(document.body);
  $(window).scroll(function () {
   $("span").css("display", "inline").fadeOut("slow");
  });
```

resize(handler(eventObject))
Bind an event handler to the "resize" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('resize', handler)` in the first and second variations, and `.trigger('resize')` in the third.

The `resize` event is sent to the `window` element when the size of the browser window changes:

```
$(window).resize(function() {
 $('#log').append('<div>Handler for .resize() called.</div>');
});
```

Now whenever the browser window's size is changed, the message is appended to one or more times, depending on the browser.

Code in a `resize` handler should never rely on the number of times the handler is called. Depending on implementation, `resize` events can be sent continuously as the resizing is in progress (the typical behavior in Internet Explorer and WebKit-based browsers such as Safari and Chrome), or only once at the end of the resize operation (the typical behavior in some other browsers such as Opera).

Example
To see the window width while (or after) it is resized, try:

```
$(window).resize(function() {
 $('body').prepend('<div>' + $(window).width() + '</div>');
});
```

error(handler(eventObject))
Bind an event handler to the "error" JavaScript event.

Arguments
handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for `.bind('error', handler)`.

The `error` event is sent to elements, such as images, that are referenced by a document and loaded by the browser. It is called if the element was not loaded correctly.

For example, consider a page with a simple image element:

```
<img alt="Book" id="book" />
```

The event handler can be bound to the image:

```
$('#book')
 .error(function() {
  alert('Handler for .error() called.')
 })
 .attr("src", "missing.png");
```

If the image cannot be loaded (for example, because it is not present at the supplied URL), the alert is displayed:

Handler for .error() called.

The event handler *must* be attached before the browser fires the error event, which is why the example sets the src attribute after attaching the handler. Also, the error event may not be correctly fired when the page is served locally; `error` relies on HTTP status codes and will generally not be triggered if the URL uses the `file:` protocol.

Note: A jQuery error event handler should not be attached to the window object. The browser fires the window's error event when a script error occurs. However, the window error event receives different arguments and has different return value requirements than conventional event handlers. Use `window.onerror` instead.

Example
To hide the "broken image" icons for IE users, you can try:

```
$("img")
 .error(function(){
  $(this).hide();
 })
 .attr("src", "missing.png");
```

Document Loading

unload(handler(eventObject))
Bind an event handler to the "unload" JavaScript event.

Arguments
handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for `.bind('unload', handler)`.

The `unload` event is sent to the `window` element when the user navigates away from the page. This could mean one of many things. The user could have clicked on a link to leave the page, or typed in a new URL in the address bar. The forward and back buttons will trigger the event. Closing the browser window will cause the event to be triggered. Even a page reload will first create an `unload` event.

The exact handling of the `unload` event has varied from version to version of browsers. For example, some versions of Firefox trigger the event when a link is followed, but not when the window is closed. In practical usage, behavior should be tested on all supported browsers, and contrasted with the proprietary `beforeunload` event.

Any `unload` event handler should be bound to the `window` object:

```
$(window).unload(function() {
 alert('Handler for .unload() called.');
});
```

After this code executes, the alert will be displayed whenever the browser leaves the current page.It is not possible to cancel the `unload` event with `.preventDefault()`. This event is available so that scripts can perform cleanup when the user leaves the page.

Example
To display an alert when a page is unloaded:

```
$(window).unload( function () { alert("Bye now!"); } );
```

load(handler(eventObject))
Bind an event handler to the "load" JavaScript event.

Arguments
handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for `.bind('load', handler)`.

The `load` event is sent to an element when it and all sub-elements have been completely loaded. This event can be sent to any element associated with a URL: images, scripts, frames, iframes, and the `window` object.

For example, consider a page with a simple image:

```
<img src="book.png" alt="Book" id="book" />
```

The event handler can be bound to the image:

```
$('#book').load(function() {
 // Handler for .load() called.
});
```

As soon as the image has been loaded, the handler is called.

In general, it is not necessary to wait for all images to be fully loaded. If code can be executed earlier, it is usually best to place it in a handler sent to the `.ready()` method.

The Ajax module also has a method named `.load()`. Which one is fired depends on the set of arguments passed.

Caveats of the `load` event when used with images

A common challenge developers attempt to solve using the `.load()` shortcut is to execute a function when an image (or collection of images) have completely loaded. There are several known caveats with this that should be noted. These are:

- It doesn't work consistently nor reliably cross-browser
- It doesn't fire correctly in WebKit if the image src is set to the same src as before
- It doesn't correctly bubble up the DOM tree
- Can cease to fire for images that already live in the browser's cache

Note: The `.live()` and `.delegate()` methods cannot be used to detect the `load` event of an iframe. The load event does not correctly bubble up the parent document and the event.target isn't set by Firefox, IE9 or Chrome, which is required to do event delegation.

Example
Run a function when the page is fully loaded including graphics.

```
$(window).load(function () {
 // run code
});
```

Example
Add the class bigImg to all images with height greater then 100 upon each image load.

```
$('img.userIcon').load(function(){
 if($(this).height() > 100) {
  $(this).addClass('bigImg');
 }
});
```

ready(handler)

Specify a function to execute when the DOM is fully loaded.

Arguments

handler - A function to execute after the DOM is ready.

While JavaScript provides the `load` event for executing code when a page is rendered, this event does not get triggered until all assets such as images have been completely received. In most cases, the script can be run as soon as the DOM hierarchy has been fully constructed. The handler passed to `.ready()` is guaranteed to be executed after the DOM is ready, so this is usually the best place to attach all other event handlers and run other jQuery code. When using scripts that rely on the value of CSS style properties, it's important to reference external stylesheets or embed style elements before referencing the scripts.

In cases where code relies on loaded assets (for example, if the dimensions of an image are required), the code should be placed in a handler for the `load` event instead.

The `.ready()` method is generally incompatible with the `<body onload="">` attribute. If `load` must be used, either do not use `.ready()` or use jQuery's `.load()` method to attach `load` event handlers to the window or to more specific items, like images.

All three of the following syntaxes are equivalent:

- `$(document).ready(handler)`
- `$().ready(handler)` (this is not recommended)
- `$(handler)`

There is also `$(document).bind("ready", handler)`. This behaves similarly to the ready method but with one exception: If the ready event has already fired and you try to `.bind("ready")` the bound handler will not be executed. Ready handlers bound this way are executed *after* any bound by the other three methods above.

The `.ready()` method can only be called on a jQuery object matching the current document, so the selector can be omitted.

The `.ready()` method is typically used with an anonymous function:

```
$(document).ready(function() {
 // Handler for .ready() called.
});
```

Which is equivalent to calling:

```
$(function() {
 // Handler for .ready() called.
});
```

If `.ready()` is called after the DOM has been initialized, the new handler passed in will be executed immediately.

Aliasing the jQuery Namespace

When using another JavaScript library, we may wish to call `$.noConflict()` to avoid namespace difficulties. When this function is called, the `$` shortcut is no longer available, forcing us to write `jQuery` each time we would normally write `$`. However, the handler passed to the `.ready()` method can take an argument, which is passed the global `jQuery` object. This means we can rename the object within the context of our `.ready()` handler without affecting other code:

```
jQuery(document).ready(function($) {
 // Code using $ as usual goes here.
});
```

Example

Display a message when the DOM is loaded.

```
$(document).ready(function () {
 $("p").text("The DOM is now loaded and can be manipulated.");
});
```

Event Handler Attachment

off(events, [selector], [handler(eventObject)])

Remove an event handler.

Arguments

events - One or more space-separated event types and optional namespaces, or just namespaces, such as "click", "keydown.myPlugin", or ".myPlugin".
selector - A selector which should match the one originally passed to `.on()` when attaching event handlers.
handler(eventObject) - A handler function previously attached for the event(s), or the special value `false`.

The `off()` method removes event handlers that were attached with .on(). See the discussion of delegated and directly bound events on that page for more information. Specific event handlers can be removed on elements by providing combinations of event names, namespaces, selectors, or handler function names. **When multiple filtering arguments are given, all of the arguments provided must match for the event handler to be removed.**

If a simple event name such as `"click"` is provided, *all* events of that type (both direct and delegated) are removed from the elements in the jQuery set. When writing code that will be used as a plugin, or simply when working with a large code base, best practice is to attach and remove events using namespaces so that the code will not inadvertently remove event handlers attached by other code. All events of all types in a specific namespace can be removed from an element by providing just a namespace, such as `".myPlugin"`. At minimum, either a namespace or event name must be provided.

To remove specific delegated event handlers, provide a `selector` argument. The selector string must exactly match the one passed to `.on()` when the event handler was attached. To remove all delegated events from an element without removing non-delegated events, use the special value `"**"`.

A handler can also be removed by specifying the function name in the `handler` argument. When jQuery attaches an event handler, it assigns a unique id to the handler function. Handlers proxied by jQuery.proxy() or a similar mechanism will all have the same unique id (the proxy function), so passing proxied handlers to `.off` may remove more handlers than intended. In those situations it is better to attach and remove event handlers using namespaces.

As with `.on()`, you can pass an `events-map` argument instead of specifying the `events` and `handler` as separate arguments. The keys are events and/or namespaces; the values are handler functions or the special value `false`.

Example

Add and remove event handlers on the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("body").on("click", "#theone", aClick)
  .find("#theone").text("Can Click!");
});
$("#unbind").click(function () {
 $("body").off("click", "#theone", aClick)
  .find("#theone").text("Does nothing...");
});
```

Example

Remove all event handlers from all paragraphs:

```
$("p").off()
```

Example

Remove all delegated click handlers from all paragraphs:

```
$("p").off( "click", "**" )
```

Example

Remove just one previously bound handler by passing it as the third argument:

```
var foo = function () {
 // code to handle some kind of event
```

```
};

// ... now foo will be called when paragraphs are clicked ...
$("body").on("click", "p", foo);


// ... foo will no longer be called.
$("body").off("click", "p", foo);
```

Example
Unbind all delegated event handlers by their namespace:

```
var validate = function () {
 // code to validate form entries
};

// delegate events under the ".validator" namespace
$("form").on("click.validator", "button", validate);

$("form").on("keypress.validator", "input[type='text']", validate);

// remove event handlers in the ".validator" namespace

$("form").off(".validator");
```

on(events, [selector], [data], handler(eventObject))
Attach an event handler function for one or more events to the selected elements.

Arguments
events - One or more space-separated event types and optional namespaces, such as "click" or "keydown.myPlugin".
selector - A selector string to filter the descendants of the selected elements that trigger the event. If the selector is `null` or omitted, the event is always triggered when it reaches the selected element.
data - Data to be passed to the handler in event.data when an event is triggered.
handler(eventObject) - A function to execute when the event is triggered. The value `false` is also allowed as a shorthand for a function that simply does `return false`.

The `.on()` method attaches event handlers to the currently selected set of elements in the jQuery object. As of jQuery 1.7, the `.on()` method provides all functionality required for attaching event handlers. For help in converting from older jQuery event methods, see .bind(), .delegate(), and .live(). To remove events bound with `.on()`, see .off(). To attach an event that runs only once and then removes itself, see .one()

Event names and namespaces

Any event names can be used for the `events` argument. jQuery will pass through the browser's standard JavaScript event types, calling the `handler` function when the browser generates events due to user actions such as `click`. In addition, the .trigger() method can trigger both standard browser event names and custom event names to call attached handlers.

An event name can be qualified by *event namespaces* that simplify removing or triggering the event. For example, `"click.myPlugin.simple"` defines both the myPlugin and simple namespaces for this particular click event. A click event handler attached via that string could be removed with `.off("click.myPlugin")` or `.off("click.simple")` without disturbing other click handlers attached to the elements. Namespaces are similar to CSS classes in that they are not hierarchical; only one name needs to match. Namespaces beginning with an underscore are reserved for jQuery's use.

In the second form of `.on()`, the `events-map` argument is a JavaScript Object, or "map". The keys are strings in the same form as the `events` argument with space-separated event type names and optional namespaces. The value for each key is a function (or `false` value) that is used as the `handler` instead of the final argument to the method. In other respects, the two forms are identical in their behavior as described below.

Direct and delegated events

The majority of browser events *bubble*, or *propagate*, from the deepest, innermost element (the **event target**) in the document where they occur all the way up to the body and the `document` element. In Internet Explorer 8 and lower, a few events such as `change` and `submit` do not natively bubble

but jQuery patches these to bubble and create consistent cross-browser behavior.

If `selector` is omitted or is null, the event handler is referred to as *direct* or *directly-bound*. The handler is called every time an event occurs on the selected elements, whether it occurs directly on the element or bubbles from a descendant (inner) element.

When a `selector` is provided, the event handler is referred to as *delegated*. The handler is not called when the event occurs directly on the bound element, but only for descendants (inner elements) that match the selector. jQuery bubbles the event from the event target up to the element where the handler is attached (i.e., innermost to outermost element) and runs the handler for any elements along that path matching the selector.

Event handlers are bound only to the currently selected elements; they must exist on the page at the time your code makes the call to `.on()`. To ensure the elements are present and can be selected, perform event binding inside a document ready handler for elements that are in the HTML markup on the page. If new HTML is being injected into the page, select the elements and attach event handlers *after* the new HTML is placed into the page. Or, use delegated events to attach an event handler, as described next.

Delegated events have the advantage that they can process events from *descendant elements* that are added to the document at a later time. By picking an element that is guaranteed to be present at the time the delegated event handler is attached, you can use delegated events to avoid the need to frequently attach and remove event handlers. This element could be the container element of a view in a Model-View-Controller design, for example, or `document` if the event handler wants to monitor all bubbling events in the document. The `document` element is available in the `head` of the document before loading any other HTML, so it is safe to attach events there without waiting for the document to be ready.

In addition to their ability to handle events on descendant elements not yet created, another advantage of delegated events is their potential for much lower overhead when many elements must be monitored. On a data table with 1,000 rows in its `tbody`, this example attaches a handler to 1,000 elements:

```
$("#dataTable tbody tr").on("click", function(event){
alert($(this).text());
});
```

A delegated-events approach attaches an event handler to only one element, the tbody, and the event only needs to bubble up one level (from the clicked `tr` to `tbody`):

```
$("#dataTable tbody").on("click", "tr", function(event){
alert($(this).text());
});
```

The event handler and its environment

The `handler` argument is a function (or the value `false`, see below), and is required unless the `events-map` form is used. You can provide an anonymous handler function at the point of the `.on()` call, as the examples have done above, or declare a named function and pass its name:

```
function notify() { alert("clicked"); }
$("button").on("click", notify);
```

When the browser triggers an event or other JavaScript calls jQuery's `.trigger()` method, jQuery passes the handler an event object it can use to analyze and change the status of the event. This object is a *normalized subset* of data provided by the browser; the browser's unmodified native event object is available in `event.originalEvent`. For example, event.type contains the event name (e.g., "resize") and event.target indicates the deepest (innermost) element where the event occurred.

By default, most events bubble up from the original event target to the `document` element. At each element along the way, jQuery calls any matching event handlers that have been attached. A handler can prevent the event from bubbling further up the document tree (and thus prevent handlers on those elements from running) by calling `event.stopPropagation()`. Any other handlers attached on the current element *will* run however. To prevent that, call `event.stopImmediatePropagation()`. (Event handlers bound to an element are called in the same order that they were bound.)

Similarly, a handler can call `event.preventDefault()` to cancel any default action that the browser may have for this event; for example, the default action on a `click` event is to follow the link. Not all browser events have default actions, and not all default actions can be canceled. See the

W3C Events Specification for details.

Returning `false` from an event handler will automatically call `event.stopPropagation()` and `event.preventDefault()`. A `false` value can also be passed for the `handler` as a shorthand for `function(){ return false; }`. So, `$("a.disabled").on("click", false);` attaches an event handler to all links with class "disabled" that prevents them from being followed when they are clicked and also stops the event from bubbling.

When jQuery calls a handler, the `this` keyword is a reference to the element where the event is being delivered; for directly bound events this is the element where the event was attached and for delegated events this is an element matching `selector`. (Note that `this` may not be equal to `event.target` if the event has bubbled from a descendant element.) To create a jQuery object from the element so that it can be used with jQuery methods, use `$(this)`.

Passing data to the handler

If a `data` argument is provided to `.on()` and is not `null` or `undefined`, it is passed to the handler in the event.data property each time an event is triggered. The `data` argument can be any type, but if a string is used the `selector` must either be provided or explicitly passed as `null` so that the data is not mistaken for a selector. Best practice is to use an object (map) so that multiple values can be passed as properties.

As of jQuery 1.4, the same event handler can be bound to an element multiple times. This is especially useful when the `event.data` feature is being used, or when other unique data resides in a closure around the event handler function. For example:

```
function greet(event) { alert("Hello "+event.data.name); }
$("button").on("click", { name: "Karl" }, greet);
$("button").on("click", { name: "Addy" }, greet);
```

The above code will generate two different alerts when the button is clicked.

As an alternative or in addition to the `data` argument provided to the `.on()` method, you can also pass data to an event handler using a second argument to .trigger() or .triggerHandler().

Event performance

In most cases, an event such as `click` occurs infrequently and performance is not a significant concern. However, high frequency events such as `mousemove` or `scroll` can fire dozens of times per second, and in those cases it becomes more important to use events judiciously. Performance can be increased by reducing the amount of work done in the handler itself, caching information needed by the handler rather than recalculating it, or by rate-limiting the number of actual page updates using `setTimeout`.

Attaching many delegated event handlers near the top of the document tree can degrade performance. Each time the event occurs, jQuery must compare all selectors of all attached events of that type to every element in the path from the event target up to the top of the document. For best performance, attach delegated events at a document location as close as possible to the target elements. Avoid excessive use of `document` or `document.body` for delegated events on large documents.

jQuery can process simple selectors of the form `tag#id.class` very quickly when they are used to filter delegated events. So, `"#myForm"`, `"a.external"`, and `"button"` are all fast selectors. Delegated events that use more complex selectors, particularly hierarchical ones, can be several times slower--although they are still fast enough for most applications. Hierarchical selectors can often be avoided simply by attaching the handler to a more appropriate point in the document. For example, instead of `$("body").on("click", "#commentForm .addNew", addComment)` use `$("#commentForm").on("click", ".addNew", addComment)`.

Additional notes

There are shorthand methods for some events such as .click() that can be used to attach or trigger event handlers. For a complete list of shorthand methods, see the events category.

Although strongly discouraged for new code, you may see the pseudo-event-name `"hover"` used as a shorthand for the string `"mouseenter mouseleave"`. It attaches a *single event handler* for those two events, and the handler must examine `event.type` to determine whether the event is `mouseenter` or `mouseleave`. Do not confuse the "hover" pseudo-event-name with the .hover() method, which accepts *one or two* functions.

jQuery's event system requires that a DOM element allow attaching data via a property on the element, so that events can be tracked and delivered.

The `object`, `embed`, and `applet` elements cannot attach data, and therefore cannot have jQuery events bound to them.

The `focus` and `blur` events are specified by the W3C to not bubble, but jQuery defines cross-browser `focusin` and `focusout` events that do bubble. When `focus` and `blur` are used to attach delegated event handlers, jQuery maps the names and delivers them as `focusin` and `focusout` respectively. For consistency and clarity, use the bubbling event type names.

jQuery also specifically prevents right and middle clicks from bubbling as they don't occur on the element being clicked. Should working with the middle click be required, the `mousedown` or `mouseup` events should be used with `.on()` instead.

In all browsers, the `load` event does not bubble. In Internet Explorer 8 and lower, the `paste` and `reset` events do not bubble. Such events are not supported for use with delegation, but they *can* be used when the event handler is directly attached to the element generating the event.

The `error` event on the `window` object uses nonstandard arguments and return value conventions, so it is not supported by jQuery. Instead, assign a handler function directly to the `window.onerror` property.

Example
Display a paragraph's text in an alert when it is clicked:

```
$("p").on("click", function(){
alert( $(this).text() );
});
```

Example
Pass data to the event handler, which is specified here by name:

```
function myHandler(event) {
alert(event.data.foo);
}
$("p").on("click", {foo: "bar"}, myHandler)
```

Example
Cancel a form submit action and prevent the event from bubbling up by returning `false`:

```
$("form").on("submit", false)
```

Example
Cancel only the default action by using .preventDefault().

```
$("form").on("submit", function(event) {
 event.preventDefault();
});
```

Example
Stop submit events from bubbling without preventing form submit, using .stopPropagation().

```
$("form").on("submit", function(event) {
 event.stopPropagation();
});
```

Example
Attach and trigger custom (non-browser) events.

```
$("p").on("myCustomEvent", function(e, myName, myValue){
 $(this).text(myName + ", hi there!");
 $("span").stop().css("opacity", 1)
  .text("myName = " + myName)
  .fadeIn(30).fadeOut(1000);
});
$("button").click(function () {
 $("p").trigger("myCustomEvent", [ "John" ]);
});
```

Example

Attach multiple event handlers simultaneously using a map.

```
$("div.test").on({
 click: function(){
  $(this).toggleClass("active");
 },
 mouseenter: function(){
  $(this).addClass("inside");
 },
 mouseleave: function(){
  $(this).removeClass("inside");
 }
});
```

Example
Click any paragraph to add another after it. Note that .on() allows a click event on any paragraph--even new ones--since the event is handled by the ever-present body element after it bubbles to there.

```
var count = 0;
  $("body").on("click", "p", function(){
   $(this).after("<p>Another paragraph! "+(++count)+"</p>");
  });
```

Example
Display each paragraph's text in an alert box whenever it is clicked:

```
$("body").on("click", "p", function(){
 alert( $(this).text() );
});
```

Example
Cancel a link's default action using the preventDefault method.

```
$("body").on("click", "a", function(event){
 event.preventDefault();
});
```

undelegate()
Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

The `.undelegate()` method is a way of removing event handlers that have been bound using .delegate(). **As of jQuery 1.7**, the .on() and .off() methods are preferred for attaching and removing event handlers.

Example
Can bind and unbind events to the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("body").delegate("#theone", "click", aClick)
  .find("#theone").text("Can Click!");
});
$("#unbind").click(function () {
 $("body").undelegate("#theone", "click", aClick)
  .find("#theone").text("Does nothing...");
});
```

Example
To unbind all delegated events from all paragraphs, write:

```
$("p").undelegate()
```

Example
To unbind all delegated click events from all paragraphs, write:

```
$("p").undelegate( "click" )
```

Example
To undelegate just one previously bound handler, pass the function in as the third argument:

```
var foo = function () {
 // code to handle some kind of event
};

// ... now foo will be called when paragraphs are clicked ...
$("body").delegate("p", "click", foo);


// ... foo will no longer be called.
$("body").undelegate("p", "click", foo);
```

Example
To unbind all delegated events by their namespace:

```
var foo = function () {
 // code to handle some kind of event
};

// delegate events under the ".whatever" namespace
$("form").delegate(":button", "click.whatever", foo);

$("form").delegate("input[type='text']", "keypress.whatever", foo);

// unbind all events delegated under the ".whatever" namespace

$("form").undelegate(".whatever");
```

delegate(selector, eventType, handler(eventObject))
Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

Arguments
selector - A selector to filter the elements that trigger the event.
eventType - A string containing one or more space-separated JavaScript event types, such as "click" or "keydown," or custom event names.
handler(eventObject) - A function to execute at the time the event is triggered.

As of jQuery 1.7, `.delegate()` has been superseded by the .on() method. For earlier versions, however, it remains the most effective means to use event delegation. More information on event binding and delegation is in the .on() method. In general, these are the equivalent templates for the two methods:

```
$(elements).delegate(selector, events, data, handler); // jQuery 1.4.3+
$(elements).on(events, selector, data, handler);    // jQuery 1.7+
```

For example, the following `.delegate()` code:

```
$("table").delegate("td", "click", function() {
 $(this).toggleClass("chosen");
});
```

is equivalent to the following code written using `.on()`:

```
$("table").on("click", "td", function() {
 $(this).toggleClass("chosen");
});
```

To remove events attached with `delegate()`, see the .undelegate() method.

Passing and handling event data works the same way as it does for `.on()`.

Example
Click a paragraph to add another. Note that .delegate() attaches a click event handler to all paragraphs - even new ones.

```
$("body").delegate("p", "click", function(){
   $(this).after("<p>Another paragraph!</p>");
  });
```

Example
To display each paragraph's text in an alert box whenever it is clicked:

```
$("body").delegate("p", "click", function(){
 alert( $(this).text() );
});
```

Example
To cancel a default action and prevent it from bubbling up, return false:

```
$("body").delegate("a", "click", function() { return false; })
```

Example
To cancel only the default action by using the preventDefault method.

```
$("body").delegate("a", "click", function(event){
 event.preventDefault();
});
```

Example
Can bind custom events too.

```
$("body").delegate("p", "myCustomEvent", function(e, myName, myValue){
   $(this).text("Hi there!");
   $("span").stop().css("opacity", 1)
        .text("myName = " + myName)
        .fadeIn(30).fadeOut(1000);
  });
  $("button").click(function () {
   $("p").trigger("myCustomEvent");
  });
```

jQuery.proxy(function, context)
Takes a function and returns a new one that will always have a particular context.

Arguments
function - The function whose context will be changed.
context - The object to which the context (`this`) of the function should be set.

This method is most useful for attaching event handlers to an element where the context is pointing back to a different object. Additionally, jQuery makes sure that even if you bind the function returned from `jQuery.proxy()` it will still unbind the correct function if passed the original.

Be aware, however, that jQuery's event binding subsystem assigns a unique id to each event handling function in order to track it when it is used to specify the function to be unbound. The function represented by `jQuery.proxy()` is seen as a single function by the event subsystem, even when it is used to bind different contexts. To avoid unbinding the wrong handler, use a unique event namespace for binding and unbinding (e.g., `"click.myproxy1"`) rather than specifying the proxied function during unbinding.

Example
Change the context of functions bound to a click handler using the "function, context" signature. Unbind the first handler after first click.

```
var me = {
 type: "zombie",
 test: function(event) {
  // Without proxy, `this` would refer to the event target
  // use event.target to reference that element.
  var element = event.target;
  $(element).css("background-color", "red");

  // With proxy, `this` refers to the me object encapsulating
  // this function.
  $("#log").append( "Hello " + this.type + "<br>" );
  $("#test").unbind("click", this.test);
 }
};

var you = {
 type: "person",
 test: function(event) {
  $("#log").append( this.type + " " );
 }
};

// execute you.test() in the context of the `you` object
// no matter where it is called
// i.e. the `this` keyword will refer to `you`
var youClick = $.proxy( you.test, you );


// attach click handlers to #test
$("#test")
 // this === "zombie"; handler unbound after first click
 .click( $.proxy( me.test, me ) )
 // this === "person"
 .click( youClick )
 // this === "zombie"
 .click( $.proxy( you.test, me ) )
 // this === "<button> element"
 .click( you.test );
```

Example
Enforce the context of the function using the "context, function name" signature. Unbind the handler after first click.

```
var obj = {
  name: "John",
  test: function() {
   $("#log").append( this.name );
   $("#test").unbind("click", obj.test);
  }
 };

 $("#test").click( jQuery.proxy( obj, "test" ) );
```

die()
Remove all event handlers previously attached using `.live()` from the elements.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to calling `.unbind()` with no arguments, which is used to remove all handlers attached with `.bind()`.See the discussions of `.live()` and `.unbind()` for further details.

As of jQuery 1.7, use of `.die()` (and its complementary method, `.live()`) is not recommended. Instead, use .off() to remove event handlers bound with .on()

Note: In order for .die() to function correctly, the selector used with it must match exactly the selector initially used with .live().

die(eventType, [handler])

Remove an event handler previously attached using `.live()` from the elements.

Arguments

eventType - A string containing a JavaScript event type, such as `click` or `keydown`.
handler - The function that is no longer to be executed.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to `.unbind()`, which is used to remove handlers attached with `.bind()`.See the discussions of `.live()` and `.unbind()` for further details.

Note: In order for `.die()` to function correctly, the selector used with it must match exactly the selector initially used with `.live()`.

Example

Can bind and unbind events to the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("#theone").live("click", aClick)
       .text("Can Click!");
});
$("#unbind").click(function () {
 $("#theone").die("click", aClick)
       .text("Does nothing...");
});
```

Example

To unbind all live events from all paragraphs, write:

```
$("p").die()
```

Example

To unbind all live click events from all paragraphs, write:

```
$("p").die( "click" )
```

Example

To unbind just one previously bound handler, pass the function in as the second argument:

```
var foo = function () {
// code to handle some kind of event
};

$("p").live("click", foo); // ... now foo will be called when paragraphs are clicked ...

$("p").die("click", foo); // ... foo will no longer be called.
```

live(events, handler(eventObject))

Attach an event handler for all elements which match the current selector, now and in the future.

Arguments

events - A string containing a JavaScript event type, such as "click" or "keydown." As of jQuery 1.4 the string can contain multiple, space-separated event types or custom event names.
handler(eventObject) - A function to execute at the time the event is triggered.

As of jQuery 1.7, the `.live()` method is deprecated. Use .on() to attach event handlers. Users of older versions of jQuery should use .delegate() in preference to `.live()`.

This method provides a means to attach delegated event handlers to the `document` element of a page, which simplifies the use of event handlers when content is dynamically added to a page. See the discussion of direct versus delegated events in the .on() method for more information.

Rewriting the `.live()` method in terms of its successors is straightforward; these are templates for equivalent calls for all three event attachment methods:

```
$(selector).live(events, data, handler);        // jQuery 1.3+
$(document).delegate(selector, events, data, handler); // jQuery 1.4.3+
$(document).on(events, selector, data, handler);    // jQuery 1.7+
```

The `events` argument can either be a space-separated list of event type names and optional namespaces, or an `event-map` of event names strings and handlers. The `data` argument is optional and can be omitted. For example, the following three method calls are functionally equivalent (but see below for more effective and performant ways to attach delegated event handlers):

```
$("a.offsite").live("click", function(){ alert("Goodbye!"); });        // jQuery 1.3+
$(document).delegate("a.offsite", "click", function(){ alert("Goodbye!"); }); // jQuery 1.4.3+
$(document).on("click", "a.offsite", function(){ alert("Goodbye!"); });    // jQuery 1.7+
```

Use of the `.live()` method is no longer recommended since later versions of jQuery offer better methods that do not have its drawbacks. In particular, the following issues arise with the use of `.live()`:

• jQuery attempts to retrieve the elements specified by the selector before calling the `.live()` method, which may be time-consuming on large documents.

• Chaining methods is not supported. For example, `$("a").find(".offsite, .external").live( ... );` is *not* valid and does not work as expected.

• Since all `.live()` events are attached at the `document` element, events take the longest and slowest possible path before they are handled.

• On mobile iOS (iPhone, iPad and iPod Touch) the `click` event does not bubble to the document body for most elements and cannot be used with `.live()` without applying one of the following workarounds:

• Use natively clickable elements such as a or `button`, as both of these do bubble to `document`.

• Use `.on()` or `.delegate()` attached to an element below the level of `document.body`, since mobile iOS does bubble within the body.

• Apply the CSS style `cursor:pointer` to the element that needs to bubble clicks (or a parent including `document.documentElement`). Note however, this will disable copypaste on the element and cause it to be highlighted when touched.

• Calling event.stopPropagation() in the event handler is ineffective in stopping event handlers attached lower in the document; the event has already propagated to `document`.

• The `.live()` method interacts with other event methods in ways that can be surprising, e.g., `$(document).unbind("click")` removes all click handlers attached by any call to `.live()`!

For pages still using `.live()`, this list of version-specific differences may be helpful:

• Before jQuery 1.7, to stop further handlers from executing after one bound using `.live()`, the handler must return `false`. Calling `.stopPropagation()` will not accomplish this.

• As of **jQuery 1.4** the `.live()` method supports custom events as well as *all JavaScript events that bubble*. It also supports certain events that don't bubble, including `change`, `submit`, `focus` and `blur`.

• In **jQuery 1.3.x** only the following JavaScript events could be bound: `click`, `dblclick`, `keydown`, `keypress`, `keyup`, `mousedown`, `mousemove`, `mouseout`, `mouseover`, and `mouseup`.

Example

Click a paragraph to add another. Note that .live() binds the click event to all paragraphs - even new ones.

```
$("p").live("click", function(){
 $(this).after("<p>Another paragraph!</p>");
});
```

Example

Cancel a default action and prevent it from bubbling up by returning false.

```
$("a").live("click", function() { return false; })
```

Example
Cancel only the default action by using the preventDefault method.

```
$("a").live("click", function(event){
 event.preventDefault();
});
```

Example
Bind custom events with .live().

```
$("p").live("myCustomEvent", function(e, myName, myValue) {
 $(this).text("Hi there!");
 $("span").stop().css("opacity", 1)
      .text("myName = " + myName)
      .fadeIn(30).fadeOut(1000);
});
$("button").click(function () {
 $("p").trigger("myCustomEvent");
});
```

Example
Use a map to bind multiple live event handlers. Note that .live() calls the click, mouseover, and mouseout event handlers for all paragraphs--even new ones.

```
$("p").live({
 click: function() {
  $(this).after("<p>Another paragraph!</p>");
 },
 mouseover: function() {
  $(this).addClass("over");
 },
 mouseout: function() {
  $(this).removeClass("over");
 }
});
```

triggerHandler(eventType, [extraParameters])
Execute all handlers attached to an element for an event.

Arguments
eventType - A string containing a JavaScript event type, such as `click` or `submit`.
extraParameters - An array of additional parameters to pass along to the event handler.

The `.triggerHandler()` method behaves similarly to `.trigger()`, with the following exceptions:

- The `.triggerHandler()` method does not cause the default behavior of an event to occur (such as a form submission).
- While `.trigger()` will operate on all elements matched by the jQuery object, `.triggerHandler()` only affects the first matched element.
- Events created with `.triggerHandler()` do not bubble up the DOM hierarchy; if they are not handled by the target element directly, they do nothing.
- Instead of returning the jQuery object (to allow chaining), `.triggerHandler()` returns whatever value was returned by the last handler it caused to be executed. If no handlers are triggered, it returns `undefined`

For more information on this method, see the discussion for `.trigger()`.

Example
If you called .triggerHandler() on a focus event - the browser's default focus action would not be triggered, only the event handlers bound to the focus event.

```
$("#old").click(function(){
$("input").trigger("focus");
});
$("#new").click(function(){
$("input").triggerHandler("focus");
```

```
});
$("input").focus(function(){
$("<span>Focused!</span>").appendTo("body").fadeOut(1000);
});
```

trigger(eventType, [extraParameters])

Execute all handlers and behaviors attached to the matched elements for the given event type.

Arguments

eventType - A string containing a JavaScript event type, such as `click` or `submit`.
extraParameters - Additional parameters to pass along to the event handler.

Any event handlers attached with `.bind()` or one of its shortcut methods are triggered when the corresponding event occurs. They can be fired manually, however, with the `.trigger()` method. A call to `.trigger()` executes the handlers in the same order they would be if the event were triggered naturally by the user:

```
$('#foo').bind('click', function() {
   alert($(this).text());
  });
  $('#foo').trigger('click');
```

As of jQuery 1.3, `.trigger()`ed events bubble up the DOM tree; an event handler can stop the bubbling by returning `false` from the handler or calling the .stopPropagation() method on the event object passed into the event. Although `.trigger()` simulates an event activation, complete with a synthesized event object, it does not perfectly replicate a naturally-occurring event.

To trigger handlers bound via jQuery without also triggering the native event, use .triggerHandler() instead.

When we define a custom event type using the `.bind()` method, the second argument to `.trigger()` can become useful. For example, suppose we have bound a handler for the `custom` event to our element instead of the built-in `click` event as we did above:

```
$('#foo').bind('custom', function(event, param1, param2) {
 alert(param1 + "n" + param2);
});
$('#foo').trigger('custom', ['Custom', 'Event']);
```

The event object is always passed as the first parameter to an event handler, but if additional parameters are specified during a `.trigger()` call, these parameters will be passed along to the handler as well. To pass more than one parameter, use an array as shown here. As of jQuery 1.6.2, a single parameter can be passed without using an array.

Note the difference between the extra parameters we're passing here and the `eventData` parameter to the .bind() method. Both are mechanisms for passing information to an event handler, but the `extraParameters` argument to `.trigger()` allows information to be determined at the time the event is triggered, while the `eventData` argument to `.bind()` requires the information to be already computed at the time the handler is bound.

The `.trigger()` method can be used on jQuery collections that wrap plain JavaScript objects similar to a pub/sub mechanism; any event handlers bound to the object will be called when the event is triggered. **Note:** For both plain objects and DOM objects, if a triggered event name matches the name of a property on the object, jQuery will attempt to invoke the property as a method if no event handler calls `event.preventDefault()`. If this behavior is not desired, use `.triggerHandler()` instead.

Example

Clicks to button #2 also trigger a click for button #1.

```
$("button:first").click(function () {
update($("span:first"));
});
$("button:last").click(function () {
$("button:first").trigger('click');

update($("span:last"));
});

function update(j) {
```

```
var n = parseInt(j.text(), 10);
j.text(n + 1);
}
```

Example
To submit the first form without using the submit() function, try:

```
$("form:first").trigger("submit")
```

Example
To submit the first form without using the submit() function, try:

```
var event = jQuery.Event("submit");
$("form:first").trigger(event);
if ( event.isDefaultPrevented() ) {
// Perform an action...
}
```

Example
To pass arbitrary data to an event:

```
$("p").click( function (event, a, b) {
// when a normal click fires, a and b are undefined
// for a trigger like below a refers to "foo" and b refers to "bar"

} ).trigger("click", ["foo", "bar"]);
```

Example
To pass arbitrary data through an event object:

```
var event = jQuery.Event("logged");
event.user = "foo";
event.pass = "bar";
$("body").trigger(event);
```

Example
Alternative way to pass data through an event object:

```
$("body").trigger({
type:"logged",
user:"foo",
pass:"bar"

});
```

one(events, [data], handler(eventObject))
Attach a handler to an event for the elements. The handler is executed at most once per element.

Arguments
events - A string containing one or more JavaScript event types, such as "click" or "submit," or custom event names.
data - A map of data that will be passed to the event handler.
handler(eventObject) - A function to execute at the time the event is triggered.

The first form of this method is identical to `.bind()`, except that the handler is unbound after its first invocation. The second two forms, introduced in jQuery 1.7, are identical to `.on()` except that the handler is removed after the first time the event occurs at the delegated element, whether the selector matched anything or not. For example:

```
$("#foo").one("click", function() {
 alert("This will be displayed only once.");
});
$("body").one("click", "#foo", function() {
 alert("This displays if #foo is the first thing clicked in the body.");
```

```
});
```

After the code is executed, a click on the element with ID `foo` will display the alert. Subsequent clicks will do nothing. This code is equivalent to:

```
$("#foo").bind("click", function( event ) {
 alert("This will be displayed only once.");
 $(this).unbind( event );
});
```

In other words, explicitly calling `.unbind()` from within a regularly-bound handler has exactly the same effect.

If the first argument contains more than one space-separated event types, the event handler is called *once for each event type*.

Example
Tie a one-time click to each div.

```
var n = 0;
$("div").one("click", function() {
 var index = $("div").index(this);
 $(this).css({
  borderStyle:"inset",
  cursor:"auto"
 });
 $("p").text("Div at index #" + index + " clicked." +
   " That's " + ++n + " total clicks.");
});
```

Example
To display the text of all paragraphs in an alert box the first time each of them is clicked:

```
$("p").one("click", function(){
alert( $(this).text() );
});
```

unbind([eventType], [handler(eventObject)])
Remove a previously-attached event handler from the elements.

Arguments
eventType - A string containing a JavaScript event type, such as `click` or `submit`.
handler(eventObject) - The function that is to be no longer executed.

Event handlers attached with `.bind()` can be removed with `.unbind()`. (As of jQuery 1.7, the .on() and .off() methods are preferred to attach and remove event handlers on elements.) In the simplest case, with no arguments, `.unbind()` removes all handlers attached to the elements:

```
$('#foo').unbind();
```

This version removes the handlers regardless of type. To be more precise, we can pass an event type:

```
$('#foo').unbind('click');
```

By specifying the `click` event type, only handlers for that event type will be unbound. This approach can still have negative ramifications if other scripts might be attaching behaviors to the same element, however. Robust and extensible applications typically demand the two-argument version for this reason:

```
var handler = function() {
 alert('The quick brown fox jumps over the lazy dog.');
};
$('#foo').bind('click', handler);
$('#foo').unbind('click', handler);
```

By naming the handler, we can be assured that no other functions are accidentally removed. Note that the following will *not* work:

```
$('#foo').bind('click', function() {
 alert('The quick brown fox jumps over the lazy dog.');
});

// will NOT work
$('#foo').unbind('click', function() {
 alert('The quick brown fox jumps over the lazy dog.');
});
```

Even though the two functions are identical in content, they are created separately and so JavaScript is free to keep them as distinct function objects. To unbind a particular handler, we need a reference to that function and not a different one that happens to do the same thing.

Note: Using a proxied function to unbind an event on an element will unbind all proxied functions on that element, as the same proxy function is used for all proxied events. To allow unbinding a specific event, use unique class names on the event (e.g. `click.proxy1`, `click.proxy2`) when attaching them.

Using Namespaces

Instead of maintaining references to handlers in order to unbind them, we can namespace the events and use this capability to narrow the scope of our unbinding actions. As shown in the discussion for the `.bind()` method, namespaces are defined by using a period (`.`) character when binding a handler:

```
$('#foo').bind('click.myEvents', handler);
```

When a handler is bound in this fashion, we can still unbind it the normal way:

```
$('#foo').unbind('click');
```

However, if we want to avoid affecting other handlers, we can be more specific:

```
$('#foo').unbind('click.myEvents');
```

We can also unbind all of the handlers in a namespace, regardless of event type:

```
$('#foo').unbind('.myEvents');
```

It is particularly useful to attach namespaces to event bindings when we are developing plug-ins or otherwise writing code that may interact with other event-handling code in the future.

Using the Event Object

The third form of the `.unbind()` method is used when we wish to unbind a handler from within itself. For example, suppose we wish to trigger an event handler only three times:

```
var timesClicked = 0;
$('#foo').bind('click', function(event) {
 alert('The quick brown fox jumps over the lazy dog.');
 timesClicked++;
 if (timesClicked >= 3) {
  $(this).unbind(event);
 }
});
```

The handler in this case must take a parameter, so that we can capture the event object and use it to unbind the handler after the third click. The event object contains the context necessary for `.unbind()` to know which handler to remove.This example is also an illustration of a closure. Since the handler refers to the `timesClicked` variable, which is defined outside the function, incrementing the variable has an effect even between invocations of the handler.

Example
Can bind and unbind events to the colored button.

```
function aClick() {
$("div").show().fadeOut("slow");
}
$("#bind").click(function () {
// could use .bind('click', aClick) instead but for variety...
$("#theone").click(aClick)
 .text("Can Click!");
});
$("#unbind").click(function () {
$("#theone").unbind('click', aClick)
 .text("Does nothing...");
});
```

Example
To unbind all events from all paragraphs, write:

```
$("p").unbind()
```

Example
To unbind all click events from all paragraphs, write:

```
$("p").unbind( "click" )
```

Example
To unbind just one previously bound handler, pass the function in as the second argument:

```
var foo = function () {
// code to handle some kind of event
};

$("p").bind("click", foo); // ... now foo will be called when paragraphs are clicked ...

$("p").unbind("click", foo); // ... foo will no longer be called.
```

bind(eventType, [eventData], handler(eventObject))
Attach a handler to an event for the elements.

Arguments
eventType - A string containing one or more DOM event types, such as "click" or "submit," or custom event names.
eventData - A map of data that will be passed to the event handler.
handler(eventObject) - A function to execute each time the event is triggered.

As of jQuery 1.7, the .on() method is the preferred method for attaching event handlers to a document. For earlier versions, the `.bind()` method is used for attaching an event handler directly to elements. Handlers are attached to the currently selected elements in the jQuery object, so those elements *must exist* at the point the call to `.bind()` occurs. For more flexible event binding, see the discussion of event delegation in .on() or .delegate().

Any string is legal for `eventType`; if the string is not the name of a native DOM event, then the handler is bound to a custom event. These events are never called by the browser, but may be triggered manually from other JavaScript code using `.trigger()` or `.triggerHandler()`.

If the `eventType` string contains a period (`.`) character, then the event is namespaced. The period character separates the event from its namespace. For example, in the call `.bind('click.name', handler)`, the string `click` is the event type, and the string `name` is the namespace. Namespacing allows us to unbind or trigger some events of a type without affecting others. See the discussion of `.unbind()` for more information.

There are shorthand methods for some standard browser events such as .click() that can be used to attach or trigger event handlers. For a complete list of shorthand methods, see the events category.

When an event reaches an element, all handlers bound to that event type for the element are fired. If there are multiple handlers registered, they will always execute in the order in which they were bound. After all handlers have executed, the event continues along the normal event propagation path.

A basic usage of `.bind()` is:

```
$('#foo').bind('click', function() {
 alert('User clicked on "foo."');
});
```

This code will cause the element with an ID of `foo` to respond to the `click` event. When a user clicks inside this element thereafter, the alert will be shown.

Multiple Events

Multiple event types can be bound at once by including each one separated by a space:

```
$('#foo').bind('mouseenter mouseleave', function() {
 $(this).toggleClass('entered');
});
```

The effect of this on `<div id="foo">` (when it does not initially have the "entered" class) is to add the "entered" class when the mouse enters the `<div>` and remove the class when the mouse leaves.

As of jQuery 1.4 we can bind multiple event handlers simultaneously by passing a map of event type/handler pairs:

```
$('#foo').bind({
 click: function() {
  // do something on click
 },
 mouseenter: function() {
  // do something on mouseenter
 }
});
```

Event Handlers

The `handler` parameter takes a callback function, as shown above. Within the handler, the keyword `this` refers to the DOM element to which the handler is bound. To make use of the element in jQuery, it can be passed to the normal `$()` function. For example:

```
$('#foo').bind('click', function() {
 alert($(this).text());
});
```

After this code is executed, when the user clicks inside the element with an ID of `foo`, its text contents will be shown as an alert.

As of jQuery 1.4.2 duplicate event handlers can be bound to an element instead of being discarded. This is useful when the event data feature is being used, or when other unique data resides in a closure around the event handler function.

In jQuery 1.4.3 you can now pass in `false` in place of an event handler. This will bind an event handler equivalent to: `function(){ return false; }`. This function can be removed at a later time by calling: `.unbind( eventName, false )`.

The Event object

The `handler` callback function can also take parameters. When the function is called, the event object will be passed to the first parameter.

The event object is often unnecessary and the parameter omitted, as sufficient context is usually available when the handler is bound to know exactly

what needs to be done when the handler is triggered. However, at times it becomes necessary to gather more information about the user's environment at the time the event was initiated. View the full Event Object.

Returning `false` from a handler is equivalent to calling both `.preventDefault()` and `.stopPropagation()` on the event object.

Using the event object in a handler looks like this:

```
$(document).ready(function() {
 $('#foo').bind('click', function(event) {
  alert('The mouse cursor is at ('
   + event.pageX + ', ' + event.pageY + ')');
 });
});
```

Note the parameter added to the anonymous function. This code will cause a click on the element with ID `foo` to report the page coordinates of the mouse cursor at the time of the click.

Passing Event Data

The optional `eventData` parameter is not commonly used. When provided, this argument allows us to pass additional information to the handler. One handy use of this parameter is to work around issues caused by closures. For example, suppose we have two event handlers that both refer to the same external variable:

```
var message = 'Spoon!';
$('#foo').bind('click', function() {
 alert(message);
});
message = 'Not in the face!';
$('#bar').bind('click', function() {
 alert(message);
});
```

Because the handlers are closures that both have `message` in their environment, both will display the message Not in the face! when triggered. The variable's value has changed. To sidestep this, we can pass the message in using `eventData`:

```
var message = 'Spoon!';
$('#foo').bind('click', {msg: message}, function(event) {
 alert(event.data.msg);
});
message = 'Not in the face!';
$('#bar').bind('click', {msg: message}, function(event) {
 alert(event.data.msg);
});
```

This time the variable is not referred to directly within the handlers; instead, the variable is passed in *by value* through `eventData`, which fixes the value at the time the event is bound. The first handler will now display Spoon! while the second will alert Not in the face!

Note that objects are passed to functions *by reference*, which further complicates this scenario.

If `eventData` is present, it is the second argument to the `.bind()` method; if no additional data needs to be sent to the handler, then the callback is passed as the second and final argument.

See the `.trigger()` method reference for a way to pass data to a handler at the time the event happens rather than when the handler is bound.

As of jQuery 1.4 we can no longer attach data (and thus, events) to object, embed, or applet elements because critical errors occur when attaching data to Java applets.

Note: Although demonstrated in the next example, it is inadvisable to bind handlers to both the `click` and `dblclick` events for the same element. The sequence of events triggered varies from browser to browser, with some receiving two click events before the `dblclick` and others only one.

Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

Example
Handle click and double-click for the paragraph. Note: the coordinates are window relative, so in this case relative to the demo iframe.

```
$("p").bind("click", function(event){
var str = "( " + event.pageX + ", " + event.pageY + " )";
$("span").text("Click happened! " + str);
});
$("p").bind("dblclick", function(){
$("span").text("Double-click happened in " + this.nodeName);
});
$("p").bind("mouseenter mouseleave", function(event){
$(this).toggleClass("over");
});
```

Example
To display each paragraph's text in an alert box whenever it is clicked:

```
$("p").bind("click", function(){
alert( $(this).text() );
});
```

Example
You can pass some extra data before the event handler:

```
function handler(event) {
alert(event.data.foo);
}
$("p").bind("click", {foo: "bar"}, handler)
```

Example
Cancel a default action and prevent it from bubbling up by returning `false`:

```
$("form").bind("submit", function() { return false; })
```

Example
Cancel only the default action by using the .preventDefault() method.

```
$("form").bind("submit", function(event) {
event.preventDefault();
});
```

Example
Stop an event from bubbling without preventing the default action by using the .stopPropagation() method.

```
$("form").bind("submit", function(event) {
 event.stopPropagation();
});
```

Example
Bind custom events.

```
$("p").bind("myCustomEvent", function(e, myName, myValue){
$(this).text(myName + ", hi there!");
$("span").stop().css("opacity", 1)
.text("myName = " + myName)
.fadeIn(30).fadeOut(1000);
});
$("button").click(function () {
$("p").trigger("myCustomEvent", [ "John" ]);
});
```

Example
Bind multiple events simultaneously.

```
$("div.test").bind({
 click: function(){
  $(this).addClass("active");
 },
 mouseenter: function(){
  $(this).addClass("inside");
 },
 mouseleave: function(){
  $(this).removeClass("inside");
 }
});
```

Event Object

event.delegateTarget
The element where the currently-called jQuery event handler was attached.

This property is most often useful in delegated events attached by .delegate() or .on(), where the event handler is attached at an ancestor of the element being processed. It can be used, for example, to identify and remove event handlers at the delegation point.

For non-delegated event handlers attached directly to an element, `event.delegateTarget` will always be equal to `event.currentTarget`.

Example
When a button in any box class is clicked, change the box's background color to red.

```
$(".box").on("click", "button", function(event) {
 $(event.delegateTarget).css("background-color", "red");
});
```

event.namespace
The namespace specified when the event was triggered.

This will likely be used primarily by plugin authors who wish to handle tasks differently depending on the event namespace used.

Example
Determine the event namespace used.

```
$("p").bind("test.something", function(event) {
 alert( event.namespace );
});
$("button").click(function(event) {
 $("p").trigger("test.something");
});
```

event.isImmediatePropagationStopped()
Returns whether event.stopImmediatePropagation() was ever called on this event object.

This property was introduced in DOM level 3.

Example
Checks whether event.stopImmediatePropagation() was called.

```
function immediatePropStopped(e) {
```

```
 var msg = "";
 if ( e.isImmediatePropagationStopped() ) {
  msg = "called"
 } else {
  msg = "not called";
 }
 $("#stop-log").append( "<div>" + msg + "</div>" );
}

$("button").click(function(event) {
 immediatePropStopped(event);
 event.stopImmediatePropagation();
 immediatePropStopped(event);
});
```

event.stopImmediatePropagation()

Keeps the rest of the handlers from being executed and prevents the event from bubbling up the DOM tree.

In addition to keeping any additional handlers on an element from being executed, this method also stops the bubbling by implicitly calling `event.stopPropagation()`. To simply prevent the event from bubbling to ancestor elements but allow other event handlers to execute on the same element, we can use `event.stopPropagation()` instead.

Use `event.isImmediatePropagationStopped()` to know whether this method was ever called (on that event object).

Example

Prevents other event handlers from being called.

```
$("p").click(function(event){
 event.stopImmediatePropagation();
});
$("p").click(function(event){
 // This function won't be executed
 $(this).css("background-color", "#f00");
});
$("div").click(function(event) {
 // This function will be executed
  $(this).css("background-color", "#f00");
});
```

event.isPropagationStopped()

Returns whether event.stopPropagation() was ever called on this event object.

This event method is described in the W3C DOM Level 3 specification.

Example

Checks whether event.stopPropagation() was called

```
function propStopped(e) {
 var msg = "";
 if ( e.isPropagationStopped() ) {
  msg = "called"
 } else {
  msg = "not called";
 }
 $("#stop-log").append( "<div>" + msg + "</div>" );
}

$("button").click(function(event) {
 propStopped(event);
```

```
 event.stopPropagation();
 propStopped(event);
});
```

event.stopPropagation()
Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.

We can use `event.isPropagationStopped()` to determine if this method was ever called (on that event object).

This method works for custom events triggered with trigger(), as well.

Note that this will not prevent other handlers *on the same element* from running.

Example
Kill the bubbling on the click event.

```
$("p").click(function(event){
 event.stopPropagation();
 // do something
});
```

event.isDefaultPrevented()
Returns whether event.preventDefault() was ever called on this event object.

Example
Checks whether event.preventDefault() was called.

```
$("a").click(function(event){
 alert( event.isDefaultPrevented() ); // false
 event.preventDefault();
 alert( event.isDefaultPrevented() ); // true
});
```

event.preventDefault()
If this method is called, the default action of the event will not be triggered.

For example, clicked anchors will not take the browser to a new URL. We can use `event.isDefaultPrevented()` to determine if this method has been called by an event handler that was triggered by this event.

Example
Cancel the default action (navigation) of the click.

```
$("a").click(function(event) {
 event.preventDefault();
 $('<div/>')
  .append('default ' + event.type + ' prevented')
  .appendTo('#log');
});
```

event.timeStamp
The difference in milliseconds between the time the browser created the event and January 1, 1970.

This property can be useful for profiling event performance by getting the `event.timeStamp` value at two points in the code and noting the difference. To simply determine the current time inside an event handler, use `(new Date).getTime()` instead.

Note: Due to a bug open since 2004, this value is not populated correctly in Firefox and it is not possible to know the time the event was created in that browser.

Example
Display the time since the click handler last executed.

```
var last, diff;
$('div').click(function(event) {
 if ( last ) {
  diff = event.timeStamp - last
  $('div').append('time since last event: ' + diff + '<br/>');
 } else {
  $('div').append('Click again.<br/>');
 }
 last = event.timeStamp;
});
```

event.result
The last value returned by an event handler that was triggered by this event, unless the value was `undefined`.

This property can be useful for getting previous return values of custom events.

Example
Display previous handler's return value

```
$("button").click(function(event) {
 return "hey";
});
$("button").click(function(event) {
 $("p").html( event.result );
});
```

event.which
For key or mouse events, this property indicates the specific key or button that was pressed.

The `event.which` property normalizes `event.keyCode` and `event.charCode`. It is recommended to watch `event.which` for keyboard key input. For more detail, read about event.charCode on the MDC.

`event.which` also normalizes button presses (`mousedown` and `mouseup`events), reporting `1` for left button, `2` for middle, and `3` for right. Use `event.which` instead of `event.button`.

Example
Log which key was depressed.

```
$('#whichkey').bind('keydown',function(e){
 $('#log').html(e.type + ': ' + e.which );
});
```

Example
Log which mouse button was depressed.

```
$('#whichkey').bind('mousedown',function(e){
 $('#log').html(e.type + ': ' + e.which );
});
```

event.pageY
The mouse position relative to the top edge of the document.

Example
Show the mouse position relative to the left and top edges of the document (within this iframe).

```
$(document).bind('mousemove',function(e){
      $("#log").text("e.pageX: " + e.pageX + ", e.pageY: " + e.pageY);
});
```

event.pageX
The mouse position relative to the left edge of the document.

Example
Show the mouse position relative to the left and top edges of the document (within the iframe).

```
$(document).bind('mousemove',function(e){
      $("#log").text("e.pageX: " + e.pageX + ", e.pageY: " + e.pageY);
});
```

event.currentTarget
The current DOM element within the event bubbling phase.

This property will typically be equal to the `this` of the function.

If you are using jQuery.proxy or another form of scope manipulation, `this` will be equal to whatever context you have provided, not `event.currentTarget`

Example
Alert that currentTarget matches the `this` keyword.

```
$("p").click(function(event) {
 alert( event.currentTarget === this ); // true
});
```

event.relatedTarget
The other DOM element involved in the event, if any.

For `mouseout`, indicates the element being entered; for `mouseover`, indicates the element being exited.

Example
On mouseout of anchors, alert the element type being entered.

```
$("a").mouseout(function(event) {
 alert(event.relatedTarget.nodeName); // "DIV"
});
```

event.data
An optional data map passed to an event method when the current executing handler is bound.

Example
Within a `for` loop, pass the value of `i` to the `.on()` method so that the current iteration's value is preserved.

```
var logDiv = $("#log");

/* Note: This code is for demonstration purposes only. */
for (var i = 0; i < 5; i++) {
 $("button").eq(i).on("click", {value: i}, function(event) {
```

```
  var msgs = [
   "button = " + $(this).index(),
   "event.data.value = " + event.data.value,
   "i = " + i
  ];
  logDiv.append( msgs.join(", ") + "<br>" );
 });
}
```

event.target
The DOM element that initiated the event.

The `target` property can be the element that registered for the event or a descendant of it. It is often useful to compare `event.target` to `this` in order to determine if the event is being handled due to event bubbling. This property is very useful in event delegation, when events bubble.

Example
Display the tag's name on click

```
$("body").click(function(event) {
 $("#log").html("clicked: " + event.target.nodeName);
});
```

Example
Implements a simple event delegation: The click handler is added to an unordered list, and the children of its li children are hidden. Clicking one of the li children toggles (see toggle()) their children.

```
function handler(event) {
 var $target = $(event.target);
 if( $target.is("li") ) {
  $target.children().toggle();
 }
}
$("ul").click(handler).find("ul").hide();
```

event.type
Describes the nature of the event.

Example
On all anchor clicks, alert the event type.

```
$("a").click(function(event) {
 alert(event.type); // "click"
});
```

Form Events

submit(handler(eventObject))
Bind an event handler to the "submit" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('submit', handler)` in the first variation, and `.trigger('submit')` in the third.

The `submit` event is sent to an element when the user is attempting to submit a form. It can only be attached to `<form>` elements. Forms can be submitted either by clicking an explicit `<input type="submit">`, `<input type="image">`, or `<button type="submit">`, or by pressing Enter when certain form elements have focus.

Depending on the browser, the Enter key may only cause a form submission if the form has exactly one text field, or only when there is a submit button present. The interface should not rely on a particular behavior for this key unless the issue is forced by observing the keypress event for presses of the Enter key.

For example, consider the HTML:

```
<form id="target" action="destination.html">
 <input type="text" value="Hello there" />
 <input type="submit" value="Go" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the form:

```
$('#target').submit(function() {
 alert('Handler for .submit() called.');
 return false;
});
```

Now when the form is submitted, the message is alerted. This happens prior to the actual submission, so we can cancel the submit action by calling `.preventDefault()` on the event object or by returning `false` from our handler. We can trigger the event manually when another element is clicked:

```
$('#other').click(function() {
 $('#target').submit();
});
```

After this code executes, clicks on Trigger the handler will also display the message. In addition, the default `submit` action on the form will be fired, so the form will be submitted.

The JavaScript `submit` event does not bubble in Internet Explorer. However, scripts that rely on event delegation with the `submit` event will work consistently across browsers as of jQuery 1.4, which has normalized the event's behavior.

Example
If you'd like to prevent forms from being submitted unless a flag variable is set, try:

```
$("form").submit(function() {
   if ($("input:first").val() == "correct") {
    $("span").text("Validated...").show();
    return true;
   }
   $("span").text("Not valid!").show().fadeOut(1000);
   return false;
  });
```

Example
If you'd like to prevent forms from being submitted unless a flag variable is set, try:

```
$("form").submit( function () {
 return this.some_flag_variable;
} );
```

Example
To trigger the submit event on the first form on the page, try:

```
$("form:first").submit();
```

select(handler(eventObject))
Bind an event handler to the "select" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('select', handler)` in the first two variations, and `.trigger('select')` in the third.

The `select` event is sent to an element when the user makes a text selection inside it. This event is limited to `<input type="text">` fields and `<textarea>` boxes.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the text input:

```
$('#target').select(function() {
 alert('Handler for .select() called.');
});
```

Now when any portion of the text is selected, the alert is displayed. Merely setting the location of the insertion point will not trigger the event. To trigger the event manually, apply `.select()` without an argument:

```
$('#other').click(function() {
 $('#target').select();
});
```

After this code executes, clicks on the Trigger button will also alert the message:

Handler for .select() called.

In addition, the default `select` action on the field will be fired, so the entire text field will be selected.

The method for retrieving the current selected text differs from one browser to another. A number of jQuery plug-ins offer cross-platform solutions.

Example
To do something when text in input boxes is selected:

```
$(":input").select( function () {
   $("div").text("Something was selected").show().fadeOut(1000);
  });
```

Example
To trigger the select event on all input elements, try:

```
$("input").select();
```

change(handler(eventObject))
Bind an event handler to the "change" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('change', handler)` in the first two variations, and `.trigger('change')` in the third.

The `change` event is sent to an element when its value changes. This event is limited to `<input>` elements, `<textarea>` boxes and `<select>` elements. For select boxes, checkboxes, and radio buttons, the event is fired immediately when the user makes a selection with the mouse, but for the other element types the event is deferred until the element loses focus.

For example, consider the HTML:

```
<form>
 <input class="target" type="text" value="Field 1" />
 <select class="target">
  <option value="option1" selected="selected">Option 1</option>
  <option value="option2">Option 2</option>
 </select>
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the text input and the select box:

```
$('.target').change(function() {
 alert('Handler for .change() called.');
});
```

Now when the second option is selected from the dropdown, the alert is displayed. It is also displayed if you change the text in the field and then click away. If the field loses focus without the contents having changed, though, the event is not triggered. To trigger the event manually, apply `.change()` without arguments:

```
$('#other').click(function() {
 $('.target').change();
});
```

After this code executes, clicks on Trigger the handler will also alert the message. The message will display twice, because the handler has been bound to the `change` event on both of the form elements.

As of jQuery 1.4, the `change` event bubbles in Internet Explorer, behaving consistently with the event in other modern browsers.

Example
Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.

```
$("select").change(function () {
     var str = "";
     $("select option:selected").each(function () {
        str += $(this).text() + " ";
       });
     $("div").text(str);
    })
    .change();
```

Example
To add a validity test to all text input elements:

```
$("input[type='text']").change( function() {
 // check input ($(this).val()) for validity here
});
```

blur(handler(eventObject))
Bind an event handler to the "blur" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('blur', handler)` in the first two variations, and `.trigger('blur')` in the third.

The `blur` event is sent to an element when it loses focus. Originally, this event was only applicable to form elements, such as `<input>`. In recent browsers, the domain of the event has been extended to include all element types. An element can lose focus via keyboard commands, such as the

Tab key, or by mouse clicks elsewhere on the page.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Field 1" />
 <input type="text" value="Field 2" />
</form>
<div id="other">
 Trigger the handler
</div>
The event handler can be bound to the first input field:
$('#target').blur(function() {
 alert('Handler for .blur() called.');
});
```

Now if the first field has the focus, clicking elsewhere or tabbing away from it displays the alert:

Handler for .blur() called.

To trigger the event programmatically, apply `.blur()` without an argument:

```
$('#other').click(function() {
 $('#target').blur();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `blur` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `blur` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `blur` to the `focusout` event in its event delegation methods, .live() and .delegate().

Example

To trigger the blur event on all paragraphs:

```
$("p").blur();
```

focus(handler(eventObject))

Bind an event handler to the "focus" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

• This method is a shortcut for `.bind('focus', handler)` in the first and second variations, and `.trigger('focus')` in the third.

• The `focus` event is sent to an element when it gains focus. This event is implicitly applicable to a limited set of elements, such as form elements (`<input>`, `<select>`, etc.) and links (`<a href>`). In recent browser versions, the event can be extended to include all element types by explicitly setting the element's `tabindex` property. An element can gain focus via keyboard commands, such as the Tab key, or by mouse clicks on the element.

• Elements with focus are usually highlighted in some way by the browser, for example with a dotted line surrounding the element. The focus is used to determine which element is the first to receive keyboard-related events.

Attempting to set focus to a hidden element causes an error in Internet Explorer. Take care to only use `.focus()` on elements that are visible. To run an element's focus event handlers without setting focus to the element, use `.triggerHandler("focus")` instead of `.focus()`.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Field 1" />
 <input type="text" value="Field 2" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the first input field:

```
$('#target').focus(function() {
 alert('Handler for .focus() called.');
});
```

Now clicking on the first field, or tabbing to it from another field, displays the alert:

Handler for .focus() called.

We can trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#target').focus();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `focus` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `focus` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `focus` to the `focusin` event in its event delegation methods, .live() and .delegate().

Example
Fire focus.

```
$("input").focus(function () {
     $(this).next("span").css('display','inline').fadeOut(1000);
  });
```

Example
To stop people from writing in text input boxes, try:

```
$("input[type=text]").focus(function(){
 $(this).blur();
});
```

Example
To focus on a login input box with id 'login' on page startup, try:

```
$(document).ready(function(){
 $("#login").focus();
});
```

Keyboard Events

focusout(handler(eventObject))
Bind an event handler to the "focusout" JavaScript event.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('focusout', handler)`.

The `focusout` event is sent to an element when it, or any element inside of it, loses focus. This is distinct from the blur event in that it supports detecting the loss of focus from parent elements (in other words, it supports event bubbling).

This event will likely be used together with the focusin event.

Example
Watch for a loss of focus to occur inside paragraphs and note the difference between the `focusout` count and the `blur` count.

```
var fo = 0, b = 0;
$("p").focusout(function() {
 fo++;
 $("#fo")
  .text("focusout fired: " + fo + "x");
}).blur(function() {
 b++;
 $("#b")
  .text("blur fired: " + b + "x");

});
```

focusin(handler(eventObject))

Bind an event handler to the "focusin" event.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('focusin', handler)`.

The `focusin` event is sent to an element when it, or any element inside of it, gains focus. This is distinct from the focus event in that it supports detecting the focus event on parent elements (in other words, it supports event bubbling).

This event will likely be used together with the focusout event.

Example

Watch for a focus to occur within the paragraphs on the page.

```
$("p").focusin(function() {
     $(this).find("span").css('display','inline').fadeOut(1000);
  });
```

keydown(handler(eventObject))

Bind an event handler to the "keydown" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('keydown', handler)` in the first and second variations, and `.trigger('keydown')` in the third.

The `keydown` event is sent to an element when the user first presses a key on the keyboard. It can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$('#target').keydown(function() {
 alert('Handler for .keydown() called.');
});
```

Now when the insertion point is inside the field, pressing a key displays the alert:

Handler for .keydown() called.

To trigger the event manually, apply `.keydown()` without an argument:

```
$('#other').click(function() {
 $('#target').keydown();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which key was pressed, examine the event object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the key code. This code corresponds to a key on the keyboard, including codes for special keys such as arrows. For catching actual text entry, `.keypress()` may be a better choice.

Example
Show the event object for the keydown handler when a key is pressed in the input.

```
var xTriggered = 0;
$('#target').keydown(function(event) {
 if (event.which == 13) {
   event.preventDefault();
  }
  xTriggered++;
  var msg = 'Handler for .keydown() called ' + xTriggered + ' time(s).';
 $.print(msg, 'html');
 $.print(event);
});

$('#other').click(function() {
 $('#target').keydown();
});
```

keyup(handler(eventObject))
Bind an event handler to the "keyup" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('keyup', handler)` in the first two variations, and `.trigger('keyup')` in the third.

The `keyup` event is sent to an element when the user releases a key on the keyboard. It can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$('#target').keyup(function() {
 alert('Handler for .keyup() called.');
});
```

Now when the insertion point is inside the field and a key is pressed and released, the alert is displayed:

Handler for .keyup() called.

To trigger the event manually, apply `.keyup()` without arguments:

```
$('#other').click(function() {
 $('#target').keyup();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which key was pressed, examine the event object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the key code. This code corresponds to a key on the keyboard, including codes for special keys such as arrows. For catching actual text entry, `.keypress()` may be a better choice.

Example

Show the event object for the keyup handler (using a simple $.print plugin) when a key is released in the input.

```
var xTriggered = 0;
$('#target').keyup(function(event) {
  xTriggered++;
  var msg = 'Handler for .keyup() called ' + xTriggered + ' time(s).';
 $.print(msg, 'html');
 $.print(event);
}).keydown(function(event) {
 if (event.which == 13) {
  event.preventDefault();
 }
});

$('#other').click(function() {
 $('#target').keyup();
});
```

keypress(handler(eventObject))

Bind an event handler to the "keypress" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

Note: as the `keypress` event isn't covered by any official specification, the actual behavior encountered when using it may differ across browsers, browser versions, and platforms.

This method is a shortcut for `.bind("keypress", handler)` in the first two variations, and `.trigger("keypress")` in the third.

The `keypress` event is sent to an element when the browser registers keyboard input. This is similar to the `keydown` event, except in the case of key repeats. If the user presses and holds a key, a `keydown` event is triggered once, but separate `keypress` events are triggered for each inserted character. In addition, modifier keys (such as Shift) trigger `keydown` events but not `keypress` events.

A `keypress` event handler can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <fieldset>
  <input id="target" type="text" value="Hello there" />
 </fieldset>
```

```
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$("#target").keypress(function() {
 alert("Handler for .keypress() called.");
});
```

Now when the insertion point is inside the field, pressing a key displays the alert:

Handler for .keypress() called.

The message repeats if the key is held down. To trigger the event manually, apply `.keypress()` without an argument::

```
$('#other').click(function() {
 $("#target").keypress();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which character was entered, examine the `event` object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the character code.

Note that `keydown` and `keyup` provide a code indicating which key is pressed, while `keypress` indicates which character was entered. For example, a lowercase "a" will be reported as 65 by `keydown` and `keyup`, but as 97 by `keypress`. An uppercase "A" is reported as 65 by all events. Because of this distinction, when catching special keystrokes such as arrow keys, `.keydown()` or `.keyup()` is a better choice.

Example

Show the event object when a key is pressed in the input. Note: This demo relies on a simple $.print() plugin (http://api.jquery.com/scripts/events.js) for the event object's output.

```
var xTriggered = 0;
$("#target").keypress(function(event) {
 if ( event.which == 13 ) {
   event.preventDefault();
  }
  xTriggered++;
  var msg = "Handler for .keypress() called " + xTriggered + " time(s).";
 $.print( msg, "html" );
 $.print( event );
});

$("#other").click(function() {
 $("#target").keypress();
});
```

Mouse Events

toggle(handler(eventObject), handler(eventObject), [handler(eventObject)])

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

Arguments

handler(eventObject) - A function to execute every even time the element is clicked.
handler(eventObject) - A function to execute every odd time the element is clicked.
handler(eventObject) - Additional handlers to cycle through after clicks.

Note: jQuery also provides an animation method named .toggle() that toggles the visibility of elements. Whether the animation or the event method is fired depends on the set of arguments passed.

The `.toggle()` method binds a handler for the `click` event, so the rules outlined for the triggering of `click` apply here as well.

```
For example, consider the HTML:
<div id="target">
 Click here
</div>
```

Event handlers can then be bound to the `<div>`:

```
$('#target').toggle(function() {
 alert('First handler for .toggle() called.');
}, function() {
 alert('Second handler for .toggle() called.');
});
```

As the element is clicked repeatedly, the messages alternate:

First handler for .toggle() called.Second handler for .toggle() called.First handler for .toggle() called.Second handler for .toggle() called.First handler for .toggle() called.

If more than two handlers are provided, `.toggle()` will cycle among all of them. For example, if there are three handlers, then the first handler will be called on the first click, the fourth click, the seventh click, and so on.

The `.toggle()` method is provided for convenience. It is relatively straightforward to implement the same behavior by hand, and this can be necessary if the assumptions built into `.toggle()` prove limiting. For example, `.toggle()` is not guaranteed to work correctly if applied twice to the same element. Since `.toggle()` internally uses a `click` handler to do its work, we must unbind `click` to remove a behavior attached with `.toggle()`, so other `click` handlers can be caught in the crossfire. The implementation also calls `.preventDefault()` on the event, so links will not be followed and buttons will not be clicked if `.toggle()` has been called on the element.

Example

Click to toggle highlight on the list item.

```
$("li").toggle(
   function () {
    $(this).css({"list-style-type":"disc", "color":"blue"});
   },
   function () {
    $(this).css({"list-style-type":"disc", "color":"red"});
   },
   function () {
    $(this).css({"list-style-type":"", "color":""});
   }
  );
```

Example

To toggle a style on table cells:

```
$("td").toggle(
 function () {
  $(this).addClass("selected");
 },
 function () {
  $(this).removeClass("selected");
 }
);
```

focusout(handler(eventObject))

Bind an event handler to the "focusout" JavaScript event.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('focusout', handler)`.

The `focusout` event is sent to an element when it, or any element inside of it, loses focus. This is distinct from the blur event in that it supports detecting the loss of focus from parent elements (in other words, it supports event bubbling).

This event will likely be used together with the focusin event.

Example
Watch for a loss of focus to occur inside paragraphs and note the difference between the `focusout` count and the `blur` count.

```
var fo = 0, b = 0;
$("p").focusout(function() {
 fo++;
 $("#fo")
  .text("focusout fired: " + fo + "x");
}).blur(function() {
 b++;
 $("#b")
  .text("blur fired: " + b + "x");

});
```

focusin(handler(eventObject))
Bind an event handler to the "focusin" event.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('focusin', handler)`.

The `focusin` event is sent to an element when it, or any element inside of it, gains focus. This is distinct from the focus event in that it supports detecting the focus event on parent elements (in other words, it supports event bubbling).

This event will likely be used together with the focusout event.

Example
Watch for a focus to occur within the paragraphs on the page.

```
$("p").focusin(function() {
     $(this).find("span").css('display','inline').fadeOut(1000);
  });
```

mousemove(handler(eventObject))
Bind an event handler to the "mousemove" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mousemove', handler)` in the first two variations, and `.trigger('mousemove')` in the third.

The `mousemove` event is sent to an element when the mouse pointer moves inside the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Move here
</div>
<div id="other">
```

```
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to the target:

```
$("#target").mousemove(function(event) {
 var msg = "Handler for .mousemove() called at ";
 msg += event.pageX + ", " + event.pageY;
 $("#log").append("<div>" + msg + "</div>");
});
```

Now when the mouse pointer moves within the target button, the messages are appended to :

Handler for .mousemove() called at (399, 48)Handler for .mousemove() called at (398, 46)Handler for .mousemove() called at (397, 44)Handler for .mousemove() called at (396, 42)

To trigger the event manually, apply `.mousemove()` without an argument:

```
$("#other").click(function() {
 $("#target").mousemove();
});
```

After this code executes, clicks on the Trigger button will also append the message:

Handler for .mousemove() called at (undefined, undefined)

When tracking mouse movement, you usually need to know the actual position of the mouse pointer. The event object that is passed to the handler contains some information about the mouse coordinates. Properties such as `.clientX`, `.offsetX`, and `.pageX` are available, but support for them differs between browsers. Fortunately, jQuery normalizes the `.pageX` and `.pageY` properties so that they can be used in all browsers. These properties provide the X and Y coordinates of the mouse pointer relative to the top-left corner of the document, as illustrated in the example output above.

Keep in mind that the `mousemove` event is triggered whenever the mouse pointer moves, even for a pixel. This means that hundreds of events can be generated over a very small amount of time. If the handler has to do any significant processing, or if multiple handlers for the event exist, this can be a serious performance drain on the browser. It is important, therefore, to optimize `mousemove` handlers as much as possible, and to unbind them as soon as they are no longer needed.

A common pattern is to bind the `mousemove` handler from within a `mousedown` hander, and to unbind it from a corresponding `mouseup` handler. If implementing this sequence of events, remember that the `mouseup` event might be sent to a different HTML element than the `mousemove` event was. To account for this, the `mouseup` handler should typically be bound to an element high up in the DOM tree, such as `<body>`.

Example

Show the mouse coordinates when the mouse is moved over the yellow div. Coordinates are relative to the window, which in this case is the iframe.

```
$("div").mousemove(function(e){
   var pageCoords = "( " + e.pageX + ", " + e.pageY + " )";
   var clientCoords = "( " + e.clientX + ", " + e.clientY + " )";
   $("span:first").text("( e.pageX, e.pageY ) : " + pageCoords);
   $("span:last").text("( e.clientX, e.clientY ) : " + clientCoords);
  });
```

hover(handlerIn(eventObject), handlerOut(eventObject))

Bind two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

Arguments

handlerIn(eventObject) - A function to execute when the mouse pointer enters the element.
handlerOut(eventObject) - A function to execute when the mouse pointer leaves the element.

The `.hover()` method binds handlers for both `mouseenter` and `mouseleave` events. You can use it to simply apply behavior to an element during the time the mouse is within the element.

Calling `$(selector).hover(handlerIn, handlerOut)` is shorthand for:

```
$(selector).mouseenter(handlerIn).mouseleave(handlerOut);
```

See the discussions for `.mouseenter()` and `.mouseleave()` for more details.

Example
To add a special style to list items that are being hovered over, try:

```
$("li").hover(
 function () {
  $(this).append($("<span> ***</span>"));
 },
 function () {
  $(this).find("span:last").remove();
 }
);



//li with fade class
$("li.fade").hover(function(){$(this).fadeOut(100);$(this).fadeIn(500);});
```

Example
To add a special style to table cells that are being hovered over, try:

```
$("td").hover(
 function () {
  $(this).addClass("hover");
 },
 function () {
  $(this).removeClass("hover");
 }
);
```

Example
To unbind the above example use:

```
$("td").unbind('mouseenter mouseleave');
```

hover(handlerInOut(eventObject))
Bind a single handler to the matched elements, to be executed when the mouse pointer enters or leaves the elements.

Arguments
handlerInOut(eventObject) - A function to execute when the mouse pointer enters or leaves the element.

The `.hover()` method, when passed a single function, will execute that handler for both `mouseenter` and `mouseleave` events. This allows the user to use jQuery's various toggle methods within the handler or to respond differently within the handler depending on the `event.type`.

Calling `$(selector).hover(handlerInOut)` is shorthand for:

```
$(selector).bind("mouseenter mouseleave", handlerInOut);
```

See the discussions for `.mouseenter()` and `.mouseleave()` for more details.

Example
Slide the next sibling LI up or down on hover, and toggle a class.

```
$("li")
.filter(":odd")
.hide()
 .end()
```

```
.filter(":even")
.hover(
 function () {
  $(this).toggleClass("active")
   .next().stop(true, true).slideToggle();
 }
);
```

mouseleave(handler(eventObject))

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseleave', handler)` in the first two variations, and `.trigger('mouseleave')` in the third.

The `mouseleave` JavaScript event is proprietary to Internet Explorer. Because of the event's general utility, jQuery simulates this event so that it can be used regardless of browser. This event is sent to an element when the mouse pointer leaves the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseleave(function() {
 $('#log').append('<div>Handler for .mouseleave() called.</div>');
});
```

Now when the mouse pointer moves out of the Outer`<div>`, the message is appended to `<div id="log">`. You can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseleave();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

The `mouseleave` event differs from `mouseout` in the way it handles event bubbling. If `mouseout` were used in this example, then when the mouse pointer moved out of the Inner element, the handler would be triggered. This is usually undesirable behavior. The `mouseleave` event, on the other hand, only triggers its handler when the mouse leaves the element it is bound to, not a descendant. So in this example, the handler is triggered when the mouse leaves the Outer element, but not the Inner element.

Example

Show number of times mouseout and mouseleave events are triggered.`mouseout` fires when the pointer moves out of child element as well, while `mouseleave` fires only when the pointer moves out of the bound element.

```
var i = 0;
  $("div.overout").mouseover(function(){
   $("p:first",this).text("mouse over");
```

```
  }).mouseout(function(){
   $("p:first",this).text("mouse out");
   $("p:last",this).text(++i);
  });

  var n = 0;
  $("div.enterleave").mouseenter(function(){
   $("p:first",this).text("mouse enter");
  }).mouseleave(function(){
   $("p:first",this).text("mouse leave");
   $("p:last",this).text(++n);
  });
```

mouseenter(handler(eventObject))

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseenter', handler)` in the first two variations, and `.trigger('mouseenter')` in the third.

The `mouseenter` JavaScript event is proprietary to Internet Explorer. Because of the event's general utility, jQuery simulates this event so that it can be used regardless of browser. This event is sent to an element when the mouse pointer enters the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseenter(function() {
 $('#log').append('<div>Handler for .mouseenter() called.</div>');
});
```

Now when the mouse pointer moves over the Outer`<div>`, the message is appended to `<div id="log">`. You can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseenter();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

The `mouseenter` event differs from `mouseover` in the way it handles event bubbling. If `mouseover` were used in this example, then when the mouse pointer moved over the Inner element, the handler would be triggered. This is usually undesirable behavior. The `mouseenter` event, on the other hand, only triggers its handler when the mouse enters the element it is bound to, not a descendant. So in this example, the handler is triggered when the mouse enters the Outer element, but not the Inner element.

Example

Show texts when mouseenter and mouseout event triggering.`mouseover` fires when the pointer moves into the child element as well, while

`mouseenter` fires only when the pointer moves into the bound element.

```
var i = 0;
  $("div.overout").mouseover(function(){
   $("p:first",this).text("mouse over");
   $("p:last",this).text(++i);
  }).mouseout(function(){
   $("p:first",this).text("mouse out");
  });

  var n = 0;
  $("div.enterleave").mouseenter(function(){
   $("p:first",this).text("mouse enter");
   $("p:last",this).text(++n);
  }).mouseleave(function(){
   $("p:first",this).text("mouse leave");
  });
```

mouseout(handler(eventObject))

Bind an event handler to the "mouseout" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseout', handler)` in the first two variation, and `.trigger('mouseout')` in the third.

The `mouseout` event is sent to an element when the mouse pointer leaves the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseout(function() {
 $('#log').append('Handler for .mouseout() called.');
});
```

Now when the mouse pointer moves out of the Outer , the message is appended to . To trigger the event manually, apply `.mouseout()` without an argument::

```
$('#other').click(function() {
 $('#outer').mouseout();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

This event type can cause many headaches due to event bubbling. For instance, when the mouse pointer moves out of the Inner element in this example, a `mouseout` event will be sent to that, then trickle up to Outer. This can trigger the bound `mouseout` handler at inopportune times. See the discussion for `.mouseleave()` for a useful alternative.

Example

Show the number of times mouseout and mouseleave events are triggered.`mouseout` fires when the pointer moves out of the child element as well, while `mouseleave` fires only when the pointer moves out of the bound element.

```
var i = 0;
  $("div.overout").mouseout(function(){
   $("p:first",this).text("mouse out");
   $("p:last",this).text(++i);
  }).mouseover(function(){
   $("p:first",this).text("mouse over");
  });

  var n = 0;
  $("div.enterleave").bind("mouseenter",function(){
   $("p:first",this).text("mouse enter");
  }).bind("mouseleave",function(){
   $("p:first",this).text("mouse leave");
   $("p:last",this).text(++n);
  });
```

mouseover(handler(eventObject))

Bind an event handler to the "mouseover" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseover', handler)` in the first two variations, and `.trigger('mouseover')` in the third.

The `mouseover` event is sent to an element when the mouse pointer enters the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseover(function() {
 $('#log').append('<div>Handler for .mouseover() called.</div>');
});
```

Now when the mouse pointer moves over the Outer`<div>`, the message is appended to `<div id="log">`. We can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseover();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

This event type can cause many headaches due to event bubbling. For instance, when the mouse pointer moves over the Inner element in this example, a `mouseover` event will be sent to that, then trickle up to Outer. This can trigger our bound `mouseover` handler at inopportune times. See

the discussion for .mouseenter() for a useful alternative.

Example
Show the number of times mouseover and mouseenter events are triggered.mouseover fires when the pointer moves into the child element as well, while mouseenter fires only when the pointer moves into the bound element.

```
var i = 0;
 $("div.overout").mouseover(function() {
  i += 1;
  $(this).find("span").text( "mouse over x " + i );
 }).mouseout(function(){
  $(this).find("span").text("mouse out ");
 });

 var n = 0;
 $("div.enterleave").mouseenter(function() {
  n += 1;
  $(this).find("span").text( "mouse enter x " + n );
 }).mouseleave(function() {
  $(this).find("span").text("mouse leave");
 });
```

dblclick(handler(eventObject))
Bind an event handler to the "dblclick" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for .bind('dblclick', handler) in the first two variations, and .trigger('dblclick') in the third.The dblclick event is sent to an element when the element is double-clicked. Any HTML element can receive this event.For example, consider the HTML:

```
<div id="target">
 Double-click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any :

```
$('#target').dblclick(function() {
 alert('Handler for .dblclick() called.');
});
```

Now double-clicking on this element displays the alert:

Handler for .dblclick() called.

To trigger the event manually, apply .dblclick() without an argument:

```
$('#other').click(function() {
 $('#target').dblclick();
});
```

After this code executes, (single) clicks on Trigger the handler will also alert the message.

The dblclick event is only triggered after this exact series of events:
- The mouse button is depressed while the pointer is inside the element.
- The mouse button is released while the pointer is inside the element.
- The mouse button is depressed again while the pointer is inside the element, within a time window that is system-dependent.

- The mouse button is released while the pointer is inside the element.

It is inadvisable to bind handlers to both the `click` and `dblclick` events for the same element. The sequence of events triggered varies from browser to browser, with some receiving two `click` events before the `dblclick` and others only one. Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

Example
To bind a "Hello World!" alert box the dblclick event on every paragraph on the page:

```
$("p").dblclick( function () { alert("Hello World!"); });
```

Example
Double click to toggle background color.

```
var divdbl = $("div:first");
  divdbl.dblclick(function () {
   divdbl.toggleClass('dbl');
  });
```

click(handler(eventObject))
Bind an event handler to the "click" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

In the first two variations, this method is a shortcut for `.bind("click", handler)`, as well as for `.on("click", handler)` as of jQuery 1.7. In the third variation, when `.click()` is called without arguments, it is a shortcut for `.trigger("click")`.

The `click` event is sent to an element when the mouse pointer is over the element, and the mouse button is pressed and released. Any HTML element can receive this event.

```
For example, consider the HTML:
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$("#target").click(function() {
 alert("Handler for .click() called.");
});
```

Now if we click on this element, the alert is displayed:

Handler for .click() called.

We can also trigger the event when a different element is clicked:

```
$("#other").click(function() {
 $("#target").click();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `click` event is only triggered after this exact series of events:
- The mouse button is depressed while the pointer is inside the element.
- The mouse button is released while the pointer is inside the element.

This is usually the desired sequence before taking an action. If this is not required, the `mousedown` or `mouseup` event may be more suitable.

Example
Hide paragraphs on a page when they are clicked:

```
$("p").click(function () {
   $(this).slideUp();
  });
```

Example
Trigger the click event on all of the paragraphs on the page:

```
$("p").click();
```

mouseup(handler(eventObject))
Bind an event handler to the "mouseup" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseup', handler)` in the first variation, and `.trigger('mouseup')` in the second.

The `mouseup` event is sent to an element when the mouse pointer is over the element, and the mouse button is released. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$('#target').mouseup(function() {
 alert('Handler for .mouseup() called.');
});
```

Now if we click on this element, the alert is displayed:

Handler for .mouseup() called.

We can also trigger the event when a different element is clicked:

```
$('#other').click(function() {
 $('#target').mouseup();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If the user clicks outside an element, drags onto it, and releases the button, this is still counted as a `mouseup` event. This sequence of actions is not treated as a button press in most user interfaces, so it is usually better to use the `click` event unless we know that the `mouseup` event is preferable for a particular situation.

Example
Show texts when mouseup and mousedown event triggering.

```
$("p").mouseup(function(){
   $(this).append('<span style="color:#F00;">Mouse up.</span>');
  }).mousedown(function(){
   $(this).append('<span style="color:#00F;">Mouse down.</span>');
  });
```

mousedown(handler(eventObject))

Bind an event handler to the "mousedown" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mousedown', handler)` in the first variation, and `.trigger('mousedown')` in the second.

The `mousedown` event is sent to an element when the mouse pointer is over the element, and the mouse button is pressed. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$('#target').mousedown(function() {
 alert('Handler for .mousedown() called.');
});
```

Now if we click on this element, the alert is displayed:

Handler for .mousedown() called.

We can also trigger the event when a different element is clicked:

```
$('#other').click(function() {
 $('#target').mousedown();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `mousedown` event is sent when any mouse button is clicked. To act only on specific buttons, we can use the event object's `which` property. Not all browsers support this property (Internet Explorer uses button instead), but jQuery normalizes the property so that it is safe to use in any browser. The value of `which` will be 1 for the left button, 2 for the middle button, or 3 for the right button.

This event is primarily useful for ensuring that the primary button was used to begin a drag operation; if ignored, strange results can occur when the user attempts to use a context menu. While the middle and right buttons can be detected with these properties, this is not reliable. In Opera and Safari, for example, right mouse button clicks are not detectable by default.

If the user clicks on an element, drags away from it, and releases the button, this is still counted as a `mousedown` event. This sequence of actions is treated as a "canceling" of the button press in most user interfaces, so it is usually better to use the `click` event unless we know that the `mousedown` event is preferable for a particular situation.

Example

Show texts when mouseup and mousedown event triggering.

```
$("p").mouseup(function(){
```

```
 $(this).append('<span style="color:#F00;">Mouse up.</span>');
}).mousedown(function(){
 $(this).append('<span style="color:#00F;">Mouse down.</span>');
});
```

Forms

submit(handler(eventObject))

Bind an event handler to the "submit" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('submit', handler)` in the first variation, and `.trigger('submit')` in the third.

The `submit` event is sent to an element when the user is attempting to submit a form. It can only be attached to `<form>` elements. Forms can be submitted either by clicking an explicit `<input type="submit">`, `<input type="image">`, or `<button type="submit">`, or by pressing Enter when certain form elements have focus.

Depending on the browser, the Enter key may only cause a form submission if the form has exactly one text field, or only when there is a submit button present. The interface should not rely on a particular behavior for this key unless the issue is forced by observing the keypress event for presses of the Enter key.

For example, consider the HTML:

```
<form id="target" action="destination.html">
 <input type="text" value="Hello there" />
 <input type="submit" value="Go" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the form:

```
$('#target').submit(function() {
 alert('Handler for .submit() called.');
 return false;
});
```

Now when the form is submitted, the message is alerted. This happens prior to the actual submission, so we can cancel the submit action by calling `.preventDefault()` on the event object or by returning `false` from our handler. We can trigger the event manually when another element is clicked:

```
$('#other').click(function() {
 $('#target').submit();
});
```

After this code executes, clicks on Trigger the handler will also display the message. In addition, the default `submit` action on the form will be fired, so the form will be submitted.

The JavaScript `submit` event does not bubble in Internet Explorer. However, scripts that rely on event delegation with the `submit` event will work consistently across browsers as of jQuery 1.4, which has normalized the event's behavior.

Example

If you'd like to prevent forms from being submitted unless a flag variable is set, try:

```
$("form").submit(function() {
   if ($("input:first").val() == "correct") {
    $("span").text("Validated...").show();
    return true;
   }
   $("span").text("Not valid!").show().fadeOut(1000);
   return false;
  });
```

Example
If you'd like to prevent forms from being submitted unless a flag variable is set, try:

```
$("form").submit( function () {
 return this.some_flag_variable;
} );
```

Example
To trigger the submit event on the first form on the page, try:

```
$("form:first").submit();
```

select(handler(eventObject))
Bind an event handler to the "select" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('select', handler)` in the first two variations, and `.trigger('select')` in the third.

The `select` event is sent to an element when the user makes a text selection inside it. This event is limited to `<input type="text">` fields and `<textarea>` boxes.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the text input:

```
$('#target').select(function() {
 alert('Handler for .select() called.');
});
```

Now when any portion of the text is selected, the alert is displayed. Merely setting the location of the insertion point will not trigger the event. To trigger the event manually, apply `.select()` without an argument:

```
$('#other').click(function() {
 $('#target').select();
});
```

After this code executes, clicks on the Trigger button will also alert the message:

Handler for .select() called.

In addition, the default `select` action on the field will be fired, so the entire text field will be selected.

The method for retrieving the current selected text differs from one browser to another. A number of jQuery plug-ins offer cross-platform solutions.

Example
To do something when text in input boxes is selected:

```
$(":input").select( function () {
   $("div").text("Something was selected").show().fadeOut(1000);
  });
```

Example
To trigger the select event on all input elements, try:

```
$("input").select();
```

change(handler(eventObject))

Bind an event handler to the "change" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('change', handler)` in the first two variations, and `.trigger('change')` in the third.

The `change` event is sent to an element when its value changes. This event is limited to `<input>` elements, `<textarea>` boxes and `<select>` elements. For select boxes, checkboxes, and radio buttons, the event is fired immediately when the user makes a selection with the mouse, but for the other element types the event is deferred until the element loses focus.

For example, consider the HTML:

```
<form>
 <input class="target" type="text" value="Field 1" />
 <select class="target">
  <option value="option1" selected="selected">Option 1</option>
  <option value="option2">Option 2</option>
 </select>
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the text input and the select box:

```
$('.target').change(function() {
 alert('Handler for .change() called.');
});
```

Now when the second option is selected from the dropdown, the alert is displayed. It is also displayed if you change the text in the field and then click away. If the field loses focus without the contents having changed, though, the event is not triggered. To trigger the event manually, apply `.change()` without arguments:

```
$('#other').click(function() {
 $('.target').change();
});
```

After this code executes, clicks on Trigger the handler will also alert the message. The message will display twice, because the handler has been bound to the `change` event on both of the form elements.

As of jQuery 1.4, the `change` event bubbles in Internet Explorer, behaving consistently with the event in other modern browsers.

Example

Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.

```
$("select").change(function () {
     var str = "";
     $("select option:selected").each(function () {
        str += $(this).text() + " ";
       });
     $("div").text(str);
    })
    .change();
```

Example

To add a validity test to all text input elements:

```
$("input[type='text']").change( function() {
 // check input ($(this).val()) for validity here
});
```

blur(handler(eventObject))

Bind an event handler to the "blur" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('blur', handler)` in the first two variations, and `.trigger('blur')` in the third.

The `blur` event is sent to an element when it loses focus. Originally, this event was only applicable to form elements, such as `<input>`. In recent browsers, the domain of the event has been extended to include all element types. An element can lose focus via keyboard commands, such as the Tab key, or by mouse clicks elsewhere on the page.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Field 1" />
 <input type="text" value="Field 2" />
</form>
<div id="other">
 Trigger the handler
</div>
The event handler can be bound to the first input field:
$('#target').blur(function() {
 alert('Handler for .blur() called.');
});
```

Now if the first field has the focus, clicking elsewhere or tabbing away from it displays the alert:

Handler for .blur() called.

To trigger the event programmatically, apply `.blur()` without an argument:

```
$('#other').click(function() {
 $('#target').blur();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `blur` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `blur` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `blur` to the `focusout` event in its event delegation methods, .live() and .delegate().

Example

To trigger the blur event on all paragraphs:

```
$("p").blur();
```

focus(handler(eventObject))

Bind an event handler to the "focus" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

- This method is a shortcut for `.bind('focus', handler)` in the first and second variations, and `.trigger('focus')` in the third.
- The `focus` event is sent to an element when it gains focus. This event is implicitly applicable to a limited set of elements, such as form elements (`<input>`, `<select>`, etc.) and links (`<a href>`). In recent browser versions, the event can be extended to include all element types by explicitly setting the element's `tabindex` property. An element can gain focus via keyboard commands, such as the Tab key, or by mouse clicks on the element.

• Elements with focus are usually highlighted in some way by the browser, for example with a dotted line surrounding the element. The focus is used to determine which element is the first to receive keyboard-related events.

Attempting to set focus to a hidden element causes an error in Internet Explorer. Take care to only use `.focus()` on elements that are visible. To run an element's focus event handlers without setting focus to the element, use `.triggerHandler("focus")` instead of `.focus()`.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Field 1" />
 <input type="text" value="Field 2" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the first input field:

```
$('#target').focus(function() {
 alert('Handler for .focus() called.');
});
```

Now clicking on the first field, or tabbing to it from another field, displays the alert:

Handler for .focus() called.

We can trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#target').focus();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `focus` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `focus` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `focus` to the `focusin` event in its event delegation methods, .live() and .delegate().

Example

Fire focus.

```
$("input").focus(function () {
     $(this).next("span").css('display','inline').fadeOut(1000);
  });
```

Example

To stop people from writing in text input boxes, try:

```
$("input[type=text]").focus(function(){
 $(this).blur();
});
```

Example

To focus on a login input box with id 'login' on page startup, try:

```
$(document).ready(function(){
 $("#login").focus();
});
```

serializeArray()

Encode a set of form elements as an array of names and values.

The `.serializeArray()` method creates a JavaScript array of objects, ready to be encoded as a JSON string. It operates on a jQuery object representing a set of form elements. The form elements can be of several types:

```
<form>
 <div><input type="text" name="a" value="1" id="a" /></div>
 <div><input type="text" name="b" value="2" id="b" /></div>
 <div><input type="hidden" name="c" value="3" id="c" /></div>
 <div>
  <textarea name="d" rows="8" cols="40">4</textarea>
 </div>
 <div><select name="e">
  <option value="5" selected="selected">5</option>
  <option value="6">6</option>
  <option value="7">7</option>
 </select></div>
 <div>
  <input type="checkbox" name="f" value="8" id="f" />
 </div>
 <div>
  <input type="submit" name="g" value="Submit" id="g" />
 </div>
</form>
```

The `.serializeArray()` method uses the standard W3C rules for successful controls to determine which elements it should include; in particular the element cannot be disabled and must contain a `name` attribute. No submit button value is serialized since the form was not submitted using a button. Data from file select elements is not serialized.

This method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization:

```
$('form').submit(function() {
 console.log($(this).serializeArray());
 return false;
});
```

This produces the following data structure (provided that the browser supports `console.log`):

```
[
 {
  name: "a",
  value: "1"
 },
 {
  name: "b",
  value: "2"
 },
 {
  name: "c",
  value: "3"
 },
 {
  name: "d",
  value: "4"
 },
 {
  name: "e",
  value: "5"
 }
]
```

Example

Get the values from a form, iterate through them, and append them to a results display.

```
function showValues() {
   var fields = $(":input").serializeArray();
   $("#results").empty();
   jQuery.each(fields, function(i, field){
    $("#results").append(field.value + " ");
   });
  }

  $(":checkbox, :radio").click(showValues);
  $("select").change(showValues);
  showValues();
```

serialize()

Encode a set of form elements as a string for submission.

The `.serialize()` method creates a text string in standard URL-encoded notation. It operates on a jQuery object representing a set of form elements. The form elements can be of several types:

```
<form>
 <div><input type="text" name="a" value="1" id="a" /></div>
 <div><input type="text" name="b" value="2" id="b" /></div>
 <div><input type="hidden" name="c" value="3" id="c" /></div>
 <div>
  <textarea name="d" rows="8" cols="40">4</textarea>
 </div>
 <div><select name="e">
  <option value="5" selected="selected">5</option>
  <option value="6">6</option>
  <option value="7">7</option>
 </select></div>
 <div>
  <input type="checkbox" name="f" value="8" id="f" />
 </div>
 <div>
  <input type="submit" name="g" value="Submit" id="g" />
 </div>
</form>
```

The `.serialize()` method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization:

```
$('form').submit(function() {
 alert($(this).serialize());
 return false;
});
```

This produces a standard-looking query string:

```
a=1&b=2&c=3&d=4&e=5
```

Warning: selecting both the form and its children will cause duplicates in the serialized string.

Note: Only "successful controls" are serialized to the string. No submit button value is serialized since the form was not submitted using a button. For a form element's value to be included in the serialized string, the element must have a `name` attribute. Values from checkboxes and radio buttons (`inputs` of type "radio" or "checkbox") are included only if they are checked. Data from file select elements is not serialized.

Example

Serialize a form to a query string, that could be sent to a server in an Ajax request.

```
function showValues() {
   var str = $("form").serialize();
```

```
   $("#results").text(str);
  }
  $(":checkbox, :radio").click(showValues);
  $("select").change(showValues);
  showValues();
```

jQuery.param(obj)

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

Arguments

obj - An array or object to serialize.

This function is used internally to convert form element values into a serialized string representation (See .serialize() for more information).

As of jQuery 1.3, the return value of a function is used instead of the function as a String.

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. You can disable this functionality globally by setting `jQuery.ajaxSettings.traditional = true;`.

If the object passed is in an Array, it must be an array of objects in the format returned by .serializeArray()

```
[{name:"first",value:"Rick"},
{name:"last",value:"Astley"},
{name:"job",value:"Rock Star"}]
```

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

Note: Because there is no universally agreed-upon specification for param strings, it is not possible to encode complex data structures using this method in a manner that works ideally across all languages supporting such input. Until such time that there is, the `$.param` method will remain in its current form.

In jQuery 1.4, HTML5 input elements are also serialized.

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));

alert(recursiveEncoded);
alert(recursiveDecoded);
```

The values of `recursiveEncoded` and `recursiveDecoded` are alerted 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);

alert(shallowEncoded);
alert(shallowDecoded);
```

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

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

Example
Serialize a key/value object.

```
var params = { width:1680, height:1050 };
  var str = jQuery.param(params);
  $("#results").text(str);
```

Example
Serialize a few complex objects

```
// <=1.3.2:
$.param({ a: [2,3,4] }) // "a=2&a=3&a=4"
// >=1.4:
$.param({ a: [2,3,4] }) // "a[]=2&a[]=3&a[]=4"

// <=1.3.2:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a=[object+Object]&d=3&d=4&d=[object+Object]"
// >=1.4:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a[b]=1&a[c]=2&d[]=3&d[]=4&d[2][e]=5"
```

val()
Get the current value of the first element in the set of matched elements.

The `.val()` method is primarily used to get the values of form elements such as `input`, `select` and `textarea`. In the case of `<select multiple="multiple">` elements, the `.val()` method returns an array containing each selected option; if no option is selected, it returns `null`.

For selects and checkboxes, you can also use the :selected and :checked selectors to get at values, for example:

```
$('select.foo option:selected').val();  // get the value from a dropdown select
$('select.foo').val();          // get the value from a dropdown select even easier
$('input:checkbox:checked').val();    // get the value from a checked checkbox
$('input:radio[name=bar]:checked').val(); // get the value from a set of radio buttons
```

Note: At present, using `.val()` on textarea elements strips carriage return characters from the browser-reported value. When this value is sent to the server via XHR however, carriage returns are preserved (or added by browsers which do not include them in the raw value). A workaround for this issue can be achieved using a valHook as follows:

```
$.valHooks.textarea = {
    get: function( elem ) {
        return elem.value.replace( /r?n/g, "rn" );
    }
};
```

Example

Get the single value from a single select and an array of values from a multiple select and display their values.

```
function displayVals() {
   var singleValues = $("#single").val();
   var multipleValues = $("#multiple").val() || [];
   $("p").html("<b>Single:</b> " +
         singleValues +
         " <b>Multiple:</b> " +
         multipleValues.join(", "));
  }

  $("select").change(displayVals);
  displayVals();
```

Example
Find the value of an input box.

```
$("input").keyup(function () {
   var value = $(this).val();
   $("p").text(value);
  }).keyup();
```

val(value)
Set the value of each element in the set of matched elements.

Arguments
value - A string of text or an array of strings corresponding to the value of each matched element to set as selected/checked.

This method is typically used to set the values of form fields.

Passing an array of element values allows matching `<input type="checkbox">`, `<input type="radio">` and `<option>`s inside of n `<select multiple="multiple">` to be selected. In the case of `<input type="radio">`s that are part of a radio group and `<select multiple="multiple">` the other elements will be deselected.

The `.val()` method allows us to set the value by passing in a function. As of jQuery 1.4, the function is passed two arguments, the current element's index and its current value:

```
$('input:text.items').val(function( index, value ) {
 return value + ' ' + this.className;
});
```

This example appends the string " items" to the text inputs' values.

Example
Set the value of an input box.

```
$("button").click(function () {
   var text = $(this).text();
   $("input").val(text);
  });
```

Example
Use the function argument to modify the value of an input box.

```
$('input').bind('blur', function() {
  $(this).val(function( i, val ) {
   return val.toUpperCase();
  });
 });
```

Example

Set a single select, a multiple select, checkboxes and a radio button .

```
$("#single").val("Single2");
  $("#multiple").val(["Multiple2", "Multiple3"]);
  $("input").val(["check1","check2", "radio1" ]);
```

Internals

jquery
A string containing the jQuery version number.

The `.jquery` property is assigned to the jQuery prototype, commonly referred to by its alias `$.fn`. It is a string containing the version number of `jQuery`, such as "1.5.0" or "1.4.4".

Example
Determine if an object is a jQuery object

```
var a = { what: "A regular JS object" },
  b = $('body');

if ( a.jquery ) { // falsy, since it's undefined
  alert(' a is a jQuery object! ');
}

if ( b.jquery ) { // truthy, since it's a string
  alert(' b is a jQuery object! ');
}
```

Example
Get the current version of jQuery running on the page

```
alert( 'You are running jQuery version: ' + $.fn.jquery );
```

jQuery.error(message)
Takes a string and throws an exception containing it.

Arguments
message - The message to send out.

This method exists primarily for plugin developers who wish to override it and provide a better display (or more information) for the error messages.

Example
Override jQuery.error for display in Firebug.

```
jQuery.error = console.error;
```

pushStack(elements)
Add a collection of DOM elements onto the jQuery stack.

Arguments
elements - An array of elements to push onto the stack and make into a new jQuery object.

Example
Add some elements onto the jQuery stack, then pop back off again.

```
jQuery([])
  .pushStack( document.getElementsByTagName("div") )
    .remove()
  .end();
```

context
The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.

The `.live()` method for binding event handlers uses this property to determine the root element to use for its event delegation needs.

The value of this property is typically equal to `document`, as this is the default context for jQuery objects if none is supplied. The context may differ if, for example, the object was created by searching within an `<iframe>` or XML document.

Note that the context property may only apply to the elements originally selected by `jQuery()`, as it is possible for the user to add elements to the collection via methods such as `.add()` and these may have a different context.

Example

Determine the exact context used.

```
$("ul")
 .append("<li>" + $("ul").context + "</li>")
 .append("<li>" + $("ul", document.body).context.nodeName + "</li>");
```

Manipulation

Class Attribute

toggleClass(className)

Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Arguments

className - One or more class names (separated by spaces) to be toggled for each element in the matched set.

This method takes one or more class names as its parameter. In the first version, if an element in the matched set of elements already has the class, then it is removed; if an element does not have the class, then it is added. For example, we can apply `.toggleClass()` to a simple `<div>`:

```
<div class="tumble">Some text.</div>
```

The first time we apply `$('div.tumble').toggleClass('bounce')`, we get the following:

```
<div class="tumble bounce">Some text.</div>
```

The second time we apply `$('div.tumble').toggleClass('bounce')`, the `<div>` class is returned to the single `tumble` value:

```
<div class="tumble">Some text.</div>
```

Applying `.toggleClass('bounce spin')` to the same `<div>` alternates between `<div class="tumble bounce spin">` and `<div class="tumble">`.

The second version of `.toggleClass()` uses the second parameter for determining whether the class should be added or removed. If this parameter's value is `true`, then the class is added; if `false`, the class is removed. In essence, the statement:

```
$('#foo').toggleClass(className, addOrRemove);
```

is equivalent to:

```
if (addOrRemove) {
  $('#foo').addClass(className);
 }
 else {
  $('#foo').removeClass(className);
 }
```

As of jQuery 1.4, if no arguments are passed to `.toggleClass()`, all class names on the element the first time `.toggleClass()` is called will be toggled. Also as of jQuery 1.4, the class name to be toggled can be determined by passing in a function.

```
$('div.foo').toggleClass(function() {
 if ($(this).parent().is('.bar')) {
  return 'happy';
 } else {
  return 'sad';
 }
});
```

This example will toggle the `happy` class for `<div class="foo">` elements if their parent element has a class of `bar`; otherwise, it will toggle the `sad` class.

Example

Toggle the class 'highlight' when a paragraph is clicked.

```
$("p").click(function () {
   $(this).toggleClass("highlight");
  });
```

Example
Add the "highlight" class to the clicked paragraph on every third click of that paragraph, remove it every first and second click.

```
var count = 0;
$("p").each(function() {
 var $thisParagraph = $(this);
 var count = 0;
 $thisParagraph.click(function() {
  count++;
  $thisParagraph.find("span").text('clicks: ' + count);
  $thisParagraph.toggleClass("highlight", count % 3 == 0);
 });
});
```

Example
Toggle the class name(s) indicated on the buttons for each div.

```
var cls = ['', 'a', 'a b', 'a b c'];
var divs = $('div.wrap').children();
var appendClass = function() {
 divs.append(function() {
  return '<div>' + (this.className || 'none') + '</div>';
 });
};

appendClass();

$('button').bind('click', function() {
 var tc = this.className || undefined;
 divs.toggleClass(tc);
 appendClass();
});

$('a').bind('click', function(event) {
 event.preventDefault();
 divs.empty().each(function(i) {
  this.className = cls[i];
 });
 appendClass();
});
```

removeClass([className])
Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Arguments
className - One or more space-separated classes to be removed from the class attribute of each matched element.

If a class name is included as a parameter, then only that class will be removed from the set of matched elements. If no class names are specified in the parameter, all classes will be removed.

More than one class may be removed at a time, separated by a space, from the set of matched elements, like so:

```
$('p').removeClass('myClass yourClass')
```

This method is often used with `.addClass()` to switch elements' classes from one to another, like so:

```
$('p').removeClass('myClass noClass').addClass('yourClass');
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

To replace all existing classes with another class, we can use `.attr('class', 'newClass')` instead.

As of jQuery 1.4, the `.removeClass()` method allows us to indicate the class to be removed by passing in a function.

```
$('li:last').removeClass(function() {
     return $(this).prev().attr('class');
    });
```

This example removes the class name of the penultimate `<li>` from the last `<li>`.

Example
Remove the class 'blue' from the matched elements.

```
$("p:even").removeClass("blue");
```

Example
Remove the class 'blue' and 'under' from the matched elements.

```
$("p:odd").removeClass("blue under");
```

Example
Remove all the classes from the matched elements.

```
$("p:eq(1)").removeClass();
```

hasClass(className)
Determine whether any of the matched elements are assigned the given class.

Arguments
className - The class name to search for.

Elements may have more than one class assigned to them. In HTML, this is represented by separating the class names with a space:

```
<div id="mydiv" class="foo bar"></div>
```

The `.hasClass()` method will return `true` if the class is assigned to an element, even if other classes also are. For example, given the HTML above, the following will return `true`:

```
$('#mydiv').hasClass('foo')
```

As would:

```
$('#mydiv').hasClass('bar')
```

While this would return `false`:

```
$('#mydiv').hasClass('quux')
```

Example
Looks for the paragraph that contains 'selected' as a class.

```
$("div#result1").append($("p:first").hasClass("selected").toString());
$("div#result2").append($("p:last").hasClass("selected").toString());
$("div#result3").append($("p").hasClass("selected").toString());
```

addClass(className)
Adds the specified class(es) to each of the set of matched elements.

Arguments

className - One or more class names to be added to the class attribute of each matched element.

It's important to note that this method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.

More than one class may be added at a time, separated by a space, to the set of matched elements, like so:

```
$("p").addClass("myClass yourClass");
```

This method is often used with `.removeClass()` to switch elements' classes from one to another, like so:

```
$("p").removeClass("myClass noClass").addClass("yourClass");
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

As of jQuery 1.4, the `.addClass()` method's argument can receive a function.

```
$("ul li:last").addClass(function(index) {
 return "item-" + index;
});
```

Given an unordered list with five `<li>` elements, this example adds the class "item-4" to the last `<li>`.

Example
Adds the class "selected" to the matched elements.

```
$("p:last").addClass("selected");
```

Example
Adds the classes "selected" and "highlight" to the matched elements.

```
$("p:last").addClass("selected highlight");
```

Example
Pass in a function to `.addClass()` to add the "green" class to a div that already has a "red" class.

```
$("div").addClass(function(index, currentClass) {
  var addedClass;

  if ( currentClass === "red" ) {
   addedClass = "green";
   $("p").text("There is one green div");
  }

  return addedClass;
 });
```

Copying

clone([withDataAndEvents])
Create a deep copy of the set of matched elements.

Arguments
withDataAndEvents - A Boolean indicating whether event handlers should be copied along with the elements. As of jQuery 1.4, element data will be copied as well.

The `.clone()` method performs a *deep* copy of the set of matched elements, meaning that it copies the matched elements as well as all of their descendant elements and text nodes. When used in conjunction with one of the insertion methods, `.clone()` is a convenient way to duplicate elements on a page. Consider the following HTML:

```
<div class="container">
```

```
 <div class="hello">Hello</div>
 <div class="goodbye">Goodbye</div>
</div>
```

As shown in the discussion for `.append()`, normally when an element is inserted somewhere in the DOM, it is moved from its old location. So, given the code:

```
$('.hello').appendTo('.goodbye');
```

The resulting DOM structure would be:

```
<div class="container">
 <div class="goodbye">
  Goodbye
  <div class="hello">Hello</div>
 </div>
</div>
```

To prevent this and instead create a copy of the element, you could write the following:

```
$('.hello').clone().appendTo('.goodbye');
```

This would produce:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">
  Goodbye
  <div class="hello">Hello</div>
 </div>
</div>
```

Note: When using the `.clone()` method, you can modify the cloned elements or their contents before (re-)inserting them into the document.

Normally, any event handlers bound to the original element are *not* copied to the clone. The optional `withDataAndEvents` parameter allows us to change this behavior, and to instead make copies of all of the event handlers as well, bound to the new copy of the element. As of jQuery 1.4, all element data (attached by the `.data()` method) is also copied to the new copy.

However, objects and arrays within element data are not copied and will continue to be shared between the cloned element and the original element. To deep copy all data, copy each one manually:

```
var $elem = $('#elem').data( "arr": [ 1 ] ), // Original element with attached data
  $clone = $elem.clone( true )
  .data( "arr", $.extend( [], $elem.data("arr") ) ); // Deep copy to prevent data sharing
```

As of jQuery 1.5, `withDataAndEvents` can be optionally enhanced with `deepWithDataAndEvents` to copy the events and data for all children of the cloned element.

Note: Using `.clone()` has the side-effect of producing elements with duplicate `id` attributes, which are supposed to be unique. Where possible, it is recommended to avoid cloning elements with this attribute or using `class` attributes as identifiers instead.

Example

Clones all b elements (and selects the clones) and prepends them to all paragraphs.

```
$("b").clone().prependTo("p");
```

Example

When using `.clone()` to clone a collection of elements that are not attached to the DOM, their order when inserted into the DOM is not guaranteed. However, it may be possible to preserve sort order with a workaround, as demonstrated:

```
// sort order is not guaranteed here and may vary with browser
$('#copy').append($('#orig .elem')
```

```
     .clone()
     .children('a')
     .prepend('foo - ')
     .parent()
     .clone());

// correct way to approach where order is maintained
$('#copy-correct')
     .append($('#orig .elem')
     .clone()
     .children('a')
     .prepend('bar - ')
     .end());
```

DOM Insertion

DOM Insertion, Around

unwrap()

Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

The `.unwrap()` method removes the element's parent. This is effectively the inverse of the `.wrap()` method. The matched elements (and their siblings, if any) replace their parents within the DOM structure.

Example

Wrap/unwrap a div around each of the paragraphs.

```
$("button").toggle(function(){
 $("p").wrap("<div></div>");
}, function(){
 $("p").unwrap();
});
```

wrapInner(wrappingElement)

Wrap an HTML structure around the content of each element in the set of matched elements.

Arguments

wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the content of the matched elements.

The `.wrapInner()` function can take any string or object that could be passed to the `$()` factory function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. The structure will be wrapped around the content of each of the elements in the set of matched elements.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using `.wrapInner()`, we can insert an HTML structure around the content of each inner `<div>` elements like so:

```
$('.inner').wrapInner('<div class="new" />');
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around the content of each matched element:

```
<div class="container">
```

```
 <div class="inner">
  <div class="new">Hello</div>
 </div>
 <div class="inner">
  <div class="new">Goodbye</div>
 </div>
</div>
```

The second version of this method allows us to instead specify a callback function. This callback function will be called once for every matched element; it should return a DOM element, jQuery object, or HTML snippet in which to wrap the content of the corresponding element. For example:

```
$('.inner').wrapInner(function() {
 return '<div class="' + this.nodeValue + '" />';
});
```

This will cause each to have a class corresponding to the text it wraps:

```
<div class="container">
 <div class="inner">
  <div class="Hello">Hello</div>
 </div>
 <div class="inner">
  <div class="Goodbye">Goodbye</div>
 </div>
</div>
```

Note: When passing a selector string to the `.wrapInner()` function, the expected input is well formed HTML with correctly closed tags. Examples of valid input include:

```
$(elem).wrapInner("<div class='test' />");
$(elem).wrapInner("<div class='test'></div>");
$(elem).wrapInner("<div class="test"></div>");
```

Example
Selects all paragraphs and wraps a bold tag around each of its contents.

```
$("p").wrapInner("<b></b>");
```

Example
Wraps a newly created tree of objects around the inside of the body.

```
$("body").wrapInner("<div><div><p><em><b></b></em></p></div></div>");
```

Example
Selects all paragraphs and wraps a bold tag around each of its contents.

```
$("p").wrapInner(document.createElement("b"));
```

Example
Selects all paragraphs and wraps a jQuery object around each of its contents.

```
$("p").wrapInner($("<span class='red'></span>"));
```

wrapAll(wrappingElement)
Wrap an HTML structure around all elements in the set of matched elements.

Arguments
wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the matched elements.

The `.wrapAll()` function can take any string or object that could be passed to the `$()` function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. The structure will be wrapped around all of the elements in the set of matched elements, as a single group.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using `.wrapAll()`, we can insert an HTML structure around the inner `<div>` elements like so:

```
$('.inner').wrapAll('<div class="new" />');
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around all matched elements:

```
<div class="container">
 <div class="new">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
 </div>
</div>
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrapAll("<div></div>");
```

Example
Wraps a newly created tree of objects around the spans. Notice anything in between the spans gets left out like the (red text) in this example. Even the white space between spans is left out. Click View Source to see the original html.

```
$("span").wrapAll("<div><div><p><em><b></b></em></p></div></div>");
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrapAll(document.createElement("div"));
```

Example
Wrap a jQuery object double depth div around all of the paragraphs. Notice it doesn't move the object but just clones it to wrap around its target.

```
$("p").wrapAll($(".doublediv"));
```

wrap(wrappingElement)
Wrap an HTML structure around each element in the set of matched elements.

Arguments
wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the matched elements.

The `.wrap()` function can take any string or object that could be passed to the `$()` factory function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. A copy of this structure will be wrapped around each of the elements in the set of matched elements. This method returns the original set of elements for chaining purposes.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using .wrap(), we can insert an HTML structure around the inner elements like so:

```
$('.inner').wrap('<div class="new" />');
```

The new element is created on the fly and added to the DOM. The result is a new wrapped around each matched element:

```
<div class="container">
 <div class="new">
  <div class="inner">Hello</div>
 </div>
 <div class="new">
  <div class="inner">Goodbye</div>
 </div>
</div>
```

The second version of this method allows us to instead specify a callback function. This callback function will be called once for every matched element; it should return a DOM element, jQuery object, or HTML snippet in which to wrap the corresponding element. For example:

```
$('.inner').wrap(function() {
 return '<div class="' + $(this).text() + '" />';
});
```

This will cause each to have a class corresponding to the text it wraps:

```
<div class="container">
 <div class="Hello">
  <div class="inner">Hello</div>
 </div>
 <div class="Goodbye">
  <div class="inner">Goodbye</div>
 </div>
</div>
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrap("<div></div>");
```

Example
Wraps a newly created tree of objects around the spans. Notice anything in between the spans gets left out like the (red text) in this example. Even the white space between spans is left out. Click View Source to see the original html.

```
$("span").wrap("<div><div><p><em><b></b></em></p></div></div>");
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrap(document.createElement("div"));
```

Example
Wrap a jQuery object double depth div around all of the paragraphs. Notice it doesn't move the object but just clones it to wrap around its target.

```
$("p").wrap($(".doublediv"));
```

DOM Insertion, Inside

prependTo(target)
Insert every element in the set of matched elements to the beginning of the target.

Arguments
target - A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted at the beginning of the element(s) specified by

this parameter.

The `.prepend()` and `.prependTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.prepend()`, the selector expression preceding the method is the container into which the content is inserted. With `.prependTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

We can create content and insert it into several elements at once:

```
$('<p>Test</p>').prependTo('.inner');
```

Each inner `<div>` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
  <p>Test</p>
  Hello
 </div>
 <div class="inner">
  <p>Test</p>
  Goodbye
 </div>
</div>
```

We can also select an element on the page and insert it into another:

```
$('h2').prependTo($('.container'));
```

If an element selected this way is inserted elsewhere, it will be moved into the target (not cloned):

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Example

Prepends all spans to the element with the ID "foo"

```
$("span").prependTo("#foo"); // check prepend() examples
```

prepend(content, [content])

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

Arguments

content - DOM element, array of elements, HTML string, or jQuery object to insert at the beginning of each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert at the beginning of each element in the set of matched elements.

The `.prepend()` method inserts the specified content as the first child of each element in the jQuery collection (To insert it as the *last* child, use .append()).

The `.prepend()` and `.prependTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.prepend()`, the selector expression preceding the method is the container into which the content is inserted. With `.prependTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it into several elements at once:

```
$('.inner').prepend('<p>Test</p>');
```

Each `<div class="inner">` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
  <p>Test</p>
  Hello
 </div>
 <div class="inner">
  <p>Test</p>
  Goodbye
 </div>
</div>
```

You can also select an element on the page and insert it into another:

```
$('.container').prepend($('h2'));
```

If *a single element* selected this way is inserted elsewhere, it will be moved into the target (*not cloned*):

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

Important: If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Additional Arguments

Similar to other content-adding methods such as `.append()` and `.before()`, `.prepend()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` as the first three child nodes of the body:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('body').prepend($newdiv1, [newdiv2, existingdiv1]);
```

Since `.prepend()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('body').prepend($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Prepends some HTML to all paragraphs.

```
$("p").prepend("<b>Hello </b>");
```

Example
Prepends a DOM Element to all paragraphs.

```
$("p").prepend(document.createTextNode("Hello "));
```

Example
Prepends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
$("p").prepend( $("b") );
```

appendTo(target)
Insert every element in the set of matched elements to the end of the target.

Arguments
target - A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted at the end of the element(s) specified by this parameter.

The `.append()` and `.appendTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.append()`, the selector expression preceding the method is the container into which the content is inserted. With `.appendTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

We can create content and insert it into several elements at once:

```
$('<p>Test</p>').appendTo('.inner');
```

Each inner `<div>` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
  Hello
  <p>Test</p>
 </div>
 <div class="inner">
  Goodbye
  <p>Test</p>
 </div>
</div>
```

We can also select an element on the page and insert it into another:

```
$('h2').appendTo($('.container'));
```

If an element selected this way is inserted elsewhere, it will be moved into the target (not cloned):

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
 <h2>Greetings</h2>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

Example
Appends all spans to the element with the ID "foo"

```
$("span").appendTo("#foo"); // check append() examples
```

append(content, [content])
Insert content, specified by the parameter, to the end of each element in the set of matched elements.

Arguments
content - DOM element, HTML string, or jQuery object to insert at the end of each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert at the end of each element in the set of matched elements.

The `.append()` method inserts the specified content as the last child of each element in the jQuery collection (To insert it as the *first* child, use .prepend()).

The `.append()` and `.appendTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.append()`, the selector expression preceding the method is the container into which the content is inserted. With `.appendTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it into several elements at once:

```
$('.inner').append('<p>Test</p>');
```

Each inner `<div>` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
  Hello
  <p>Test</p>
 </div>
 <div class="inner">
  Goodbye
  <p>Test</p>
 </div>
</div>
```

You can also select an element on the page and insert it into another:

```
$('.container').append($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved into the target (not cloned):

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
 <h2>Greetings</h2>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.before()`, `.append()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` as the last three child nodes of the body:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('body').append($newdiv1, [newdiv2, existingdiv1]);
```

Since `.append()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('body').append($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Appends some HTML to all paragraphs.

```
$("p").append("<strong>Hello</strong>");
```

Example
Appends an Element to all paragraphs.

```
$("p").append(document.createTextNode("Hello"));
```

Example
Appends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
$("p").append( $("strong") );
```

text()
Get the combined text contents of each element in the set of matched elements, including their descendants.

Unlike the `.html()` method, `.text()` can be used in both XML and HTML documents. The result of the `.text()` method is a string containing the combined text of all matched elements. (Due to variations in the HTML parsers in different browsers, the text returned may vary in newlines and other white space.) Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
 <ul>
 <li>list item 1</li>
 <li>list <strong>item</strong> 2</li>
 </ul>
```

```
 </div>
```

The code `$('div.demo-container').text()` would produce the following result:

```
Demonstration Box list item 1 list item 2
```

The `.text()` method cannot be used on form inputs or scripts. To set or get the text value of `input` or `textarea` elements, use the .val() method. To get the value of a script element, use the .html() method.

As of jQuery 1.4, the `.text()` method returns the value of text and CDATA nodes as well as element nodes.

Example
Find the text in the first paragraph (stripping out the html), then set the html of the last paragraph to show it is just text (the red bold is gone).

```
var str = $("p:first").text();
  $("p:last").html(str);
```

text(textString)
Set the content of each element in the set of matched elements to the specified text.

Arguments
textString - A string of text to set as the content of each matched element.

Unlike the `.html()` method, `.text()` can be used in both XML and HTML documents.

We need to be aware that this method escapes the string provided as necessary so that it will render correctly in HTML. To do so, it calls the DOM method `.createTextNode()`, which replaces special characters with their HTML entity equivalents (such as `<` for <). Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
 <ul>
  <li>list item 1</li>
  <li>list <strong>item</strong> 2</li>
 </ul>
</div>
```

The code `$('div.demo-container').text('<p>This is a test.</p>');` will produce the following DOM output:

```
<div class="demo-container">
&lt;p&gt;This is a test.&lt;/p&gt;
</div>
```

It will appear on a rendered page as though the tags were exposed, like this:

```
<p>This is a test</p>
```

The `.text()` method cannot be used on input elements. For input field text, use the .val() method.

As of jQuery 1.4, the `.text()` method allows us to set the text content by passing in a function.

```
$('ul li').text(function(index) {
 return 'item number ' + (index + 1);
});
```

Given an unordered list with three `<li>` elements, this example will produce the following DOM output:

```
<ul>
 <li>item number 1</li>
 <li>item number 2</li>
```

```
 <li>item number 3</li>
</ul>
```

Example
Add text to the paragraph (notice the bold tag is escaped).

```
$("p").text("<b>Some</b> new text.");
```

html()
Get the HTML contents of the first element in the set of matched elements.

This method is not available on XML documents.

In an HTML document, `.html()` can be used to get the contents of any element. If the selector expression matches more than one element, only the first match will have its HTML content returned. Consider this code:

```
$('div.demo-container').html();
```

In order for the following `<div>`'s content to be retrieved, it would have to be the first one with `class="demo-container"` in the document:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
</div>
```

The result would look like this:

```
<div class="demo-box">Demonstration Box</div>
```

This method uses the browser's `innerHTML` property. Some browsers may not return HTML that exactly replicates the HTML source in an original document. For example, Internet Explorer sometimes leaves off the quotes around attribute values if they contain only alphanumeric characters.

Example
Click a paragraph to convert it from html to text.

```
$("p").click(function () {
   var htmlStr = $(this).html();
   $(this).text(htmlStr);
  });
```

html(htmlString)
Set the HTML contents of each element in the set of matched elements.

Arguments
htmlString - A string of HTML to set as the content of each matched element.

The `.html()` method is not available in XML documents.

When `.html()` is used to set an element's content, any content that was in that element is completely replaced by the new content. Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
</div>
```

The content of `<div class="demo-container">` can be set like this:

```
$('div.demo-container')
 .html('<p>All new content. <em>You bet!</em></p>');
```

That line of code will replace everything inside `<div class="demo-container">`:

```
<div class="demo-container">
 <p>All new content. <em>You bet!</em></p>
</div>
```

As of jQuery 1.4, the `.html()` method allows the HTML content to be set by passing in a function.

```
$('div.demo-container').html(function() {
 var emph = '<em>' + $('p').length + ' paragraphs!</em>';
 return '<p>All new content for ' + emph + '</p>';
});
```

Given a document with six paragraphs, this example will set the HTML of `<div class="demo-container">` to `<p>All new content for <em>6 paragraphs!</em></p>`.

This method uses the browser's `innerHTML` property. Some browsers may not generate a DOM that exactly replicates the HTML source provided. For example, Internet Explorer prior to version 8 will convert all `href` properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate compatibility layer.

Note: In Internet Explorer up to and including version 9, setting the text content of an HTML element may corrupt the text nodes of its children that are being removed from the document as a result of the operation. If you are keeping references to these DOM elements and need them to be unchanged, use `.empty().html(string)` instead of `.html(string)` so that the elements are removed from the document before the new string is assigned to the element.

Example
Add some html to each div.

```
$("div").html("<span class='red'>Hello <b>Again</b></span>");
```

Example
Add some html to each div then immediately do further manipulations to the inserted html.

```
$("div").html("<b>Wow!</b> Such excitement...");
  $("div b").append(document.createTextNode("!!!"))
       .css("color", "red");
```

DOM Insertion, Outside

insertBefore(target)
Insert every element in the set of matched elements before the target.

Arguments
target - A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted before the element(s) specified by this parameter.

The `.before()` and `.insertBefore()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.before()`, the selector expression preceding the method is the container before which the content is inserted. With `.insertBefore()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted before the target container.

Consider the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

We can create content and insert it before several elements at once:

```
$('<p>Test</p>').insertBefore('.inner');
```

Each inner element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <p>Test</p>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
</div>
```

We can also select an element on the page and insert it before another:

```
$('h2').insertBefore($('.container'));
```

If an element selected this way is inserted elsewhere, it will be moved before the target (not cloned):

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

Example

Inserts all paragraphs before an element with id of "foo". Same as $("#foo").before("p")

```
$("p").insertBefore("#foo"); // check before() examples
```

before(content, [content])

Insert content, specified by the parameter, before each element in the set of matched elements.

Arguments

content - HTML string, DOM element, or jQuery object to insert before each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert before each element in the set of matched elements.

The `.before()` and `.insertBefore()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.before()`, the selector expression preceding the method is the container before which the content is inserted. With `.insertBefore()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted before the target container.

Consider the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it before several elements at once:

```
$('.inner').before('<p>Test</p>');
```

Each inner element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <p>Test</p>
```

```
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
</div>
```

You can also select an element on the page and insert it before another:

```
$('.container').before($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved before the target (not cloned):

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

In jQuery 1.4, `.before()` and `.after()` will also work on disconnected DOM nodes:

```
$("<div/>").before("<p></p>");
```

The result is a jQuery set that contains a paragraph and a div (in that order).

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.after()`, `.before()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` before the first paragraph:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('p').first().before($newdiv1, [newdiv2, existingdiv1]);
```

Since `.before()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('p').first().before($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Inserts some HTML before all paragraphs.

```
$("p").before("<b>Hello</b>");
```

Example
Inserts a DOM element before all paragraphs.

```
$("p").before( document.createTextNode("Hello") );
```

Example
Inserts a jQuery object (similar to an Array of DOM Elements) before all paragraphs.

```
$("p").before( $("b") );
```

insertAfter(target)
Insert every element in the set of matched elements after the target.

Arguments

target - A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted after the element(s) specified by this parameter.

The `.after()` and `.insertAfter()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.after()`, the selector expression preceding the method is the container after which the content is inserted. With `.insertAfter()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted after the target container.

Consider the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

We can create content and insert it after several elements at once:

```
$('<p>Test</p>').insertAfter('.inner');
```

Each inner `<div>` element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
 <p>Test</p>
</div>
```

We can also select an element on the page and insert it after another:

```
$('h2').insertAfter($('.container'));
```

If an element selected this way is inserted elsewhere, it will be moved after the target (not cloned):

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
<h2>Greetings</h2>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

Example

Inserts all paragraphs after an element with id of "foo". Same as $("#foo").after("p")

```
$("p").insertAfter("#foo"); // check after() examples
```

after(content, [content])

Insert content, specified by the parameter, after each element in the set of matched elements.

Arguments

content - HTML string, DOM element, or jQuery object to insert after each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert after each element in the set of matched elements.

The `.after()` and `.insertAfter()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.after()`, the selector expression preceding the method is the container after which the content is inserted. With `.insertAfter()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is

inserted after the target container.

Using the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Content can be created and then inserted after several elements at once:

```
$('.inner').after('<p>Test</p>');
```

Each inner `<div>` element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
 <p>Test</p>
</div>
```

An element in the DOM can also be selected and inserted after another element:

```
$('.container').after($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved rather than cloned:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
<h2>Greetings</h2>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Inserting Disconnected DOM nodes

As of jQuery 1.4, `.before()` and `.after()` will also work on disconnected DOM nodes. For example, given the following code:

```
$('<div/>').after('<p></p>');
```

The result is a jQuery set containing a div and a paragraph, in that order. That set can be further manipulated, even before it is inserted in the document.

```
$('<div/>').after('<p></p>').addClass('foo')
 .filter('p').attr('id', 'bar').html('hello')
.end()
.appendTo('body');
```

This results in the following elements inserted just before the closing `</body>` tag:

```
<div class="foo"></div>
<p class="foo" id="bar">hello</p>
```

Passing a Function

As of jQuery 1.4, `.after()` supports passing a function that returns the elements to insert.

```
$('p').after(function() {
 return '<div>' + this.className + '</div>';
});
```

This example inserts a `<div>` after each paragraph, with each new `<div>` containing the class name(s) of its preceding paragraph.

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.before()`, `.after()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` after the first paragraph:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('p').first().after($newdiv1, [newdiv2, existingdiv1]);
```

Since `.after()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('p').first().after($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on the elements that are collected in the code.

Example
Inserts some HTML after all paragraphs.

```
$("p").after("<b>Hello</b>");
```

Example
Inserts a DOM element after all paragraphs.

```
$("p").after( document.createTextNode("Hello") );
```

Example
Inserts a jQuery object (similar to an Array of DOM Elements) after all paragraphs.

```
$("p").after( $("b") );
```

DOM Removal

unwrap()
Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

The `.unwrap()` method removes the element's parent. This is effectively the inverse of the `.wrap()` method. The matched elements (and their siblings, if any) replace their parents within the DOM structure.

Example
Wrap/unwrap a div around each of the paragraphs.

```
$("button").toggle(function(){
 $("p").wrap("<div></div>");
}, function(){
 $("p").unwrap();
});
```

detach([selector])
Remove the set of matched elements from the DOM.

Arguments
selector - A selector expression that filters the set of matched elements to be removed.

The `.detach()` method is the same as `.remove()`, except that `.detach()` keeps all jQuery data associated with the removed elements. This method is useful when removed elements are to be reinserted into the DOM at a later time.

Example
Detach all paragraphs from the DOM

```
$("p").click(function(){
   $(this).toggleClass("off");
  });
  var p;
  $("button").click(function(){
   if ( p ) {
    p.appendTo("body");
    p = null;
   } else {
    p = $("p").detach();
   }
  });
```

remove([selector])
Remove the set of matched elements from the DOM.

Arguments
selector - A selector expression that filters the set of matched elements to be removed.

Similar to `.empty()`, the `.remove()` method takes elements out of the DOM. Use `.remove()` when you want to remove the element itself, as well as everything inside it. In addition to the elements themselves, all bound events and jQuery data associated with the elements are removed. To remove the elements without removing data and events, use `.detach()` instead.

Consider the following HTML:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">Goodbye</div>
</div>
```

We can target any element for removal:

```
$('.hello').remove();
```

This will result in a DOM structure with the `<div>` element deleted:

```
<div class="container">
 <div class="goodbye">Goodbye</div>
</div>
```

If we had any number of nested elements inside `<div class="hello">`, they would be removed, too. Other jQuery constructs such as data or event handlers are erased as well.

We can also include a selector as an optional parameter. For example, we could rewrite the previous DOM removal code as follows:

```
$('div').remove('.hello');
```

This would result in the same DOM structure:

```
<div class="container">
```

```
 <div class="goodbye">Goodbye</div>
</div>
```

Example
Removes all paragraphs from the DOM

```
$("button").click(function () {
   $("p").remove();
  });
```

Example
Removes all paragraphs that contain "Hello" from the DOM. Analogous to doing `$("p").filter(":contains('Hello')").remove()`.

```
$("button").click(function () {
   $("p").remove(":contains('Hello')");
  });
```

empty()
Remove all child nodes of the set of matched elements from the DOM.

This method removes not only child (and other descendant) elements, but also any text within the set of matched elements. This is because, according to the DOM specification, any string of text within an element is considered a child node of that element. Consider the following HTML:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">Goodbye</div>
</div>
```

We can target any element for removal:

```
$('.hello').empty();
```

This will result in a DOM structure with the `Hello` text deleted:

```
<div class="container">
 <div class="hello"></div>
 <div class="goodbye">Goodbye</div>
</div>
```

If we had any number of nested elements inside `<div class="hello">`, they would be removed, too.

To avoid memory leaks, jQuery removes other constructs such as data and event handlers from the child elements before removing the elements themselves.

If you want to remove elements without destroying their data or event handlers (so they can be re-added later), use .detach() instead.

Example
Removes all child nodes (including text nodes) from all paragraphs

```
$("button").click(function () {
  $("p").empty();
 });
```

DOM Replacement

replaceAll(target)
Replace each target element with the set of matched elements.

Arguments

target - A selector expression indicating which element(s) to replace.

The `.replaceAll()` method is corollary to `.replaceWith()`, but with the source and target reversed. Consider this DOM structure:

```
<div class="container">
 <div class="inner first">Hello</div>
 <div class="inner second">And</div>
 <div class="inner third">Goodbye</div>
</div>
```

We can create an element, then replace other elements with it:

```
$('<h2>New heading</h2>').replaceAll('.inner');
```

This causes all of them to be replaced:

```
<div class="container">
 <h2>New heading</h2>
 <h2>New heading</h2>
 <h2>New heading</h2>
</div>
```

Or, we could select an element to use as the replacement:

```
$('.first').replaceAll('.third');
```

This results in the DOM structure:

```
<div class="container">
 <div class="inner second">And</div>
 <div class="inner first">Hello</div>
</div>
```

From this example, we can see that the selected element replaces the target by being moved from its old location, not by being cloned.

Example
Replace all the paragraphs with bold words.

```
$("<b>Paragraph. </b>").replaceAll("p"); // check replaceWith() examples
```

replaceWith(newContent)
Replace each element in the set of matched elements with the provided new content.

Arguments
newContent - The content to insert. May be an HTML string, DOM element, or jQuery object.

The `.replaceWith()` method removes content from the DOM and inserts new content in its place with a single call. Consider this DOM structure:

```
<div class="container">
 <div class="inner first">Hello</div>
 <div class="inner second">And</div>
 <div class="inner third">Goodbye</div>
</div>
```

The second inner `<div>` could be replaced with the specified HTML:

```
$('div.second').replaceWith('<h2>New heading</h2>');
```

This results in the structure:

```
<div class="container">
 <div class="inner first">Hello</div>
```

```
 <h2>New heading</h2>
 <div class="inner third">Goodbye</div>
</div>
```

All inner elements could be targeted at once:

```
$('div.inner').replaceWith('<h2>New heading</h2>');
```

This causes all of them to be replaced:

```
<div class="container">
 <h2>New heading</h2>
 <h2>New heading</h2>
 <h2>New heading</h2>
</div>
```

An element could also be selected as the replacement:

```
$('div.third').replaceWith($('.first'));
```

This results in the DOM structure:

```
<div class="container">
 <div class="inner second">And</div>
 <div class="inner first">Hello</div>
</div>
```

This example demonstrates that the selected element replaces the target by being moved from its old location, not by being cloned.

The `.replaceWith()` method, like most jQuery methods, returns the jQuery object so that other methods can be chained onto it. However, it must be noted that the *original* jQuery object is returned. This object refers to the element that has been removed from the DOM, not the new element that has replaced it.

As of jQuery 1.4, `.replaceWith()` can also work on disconnected DOM nodes. For example, with the following code, `.replaceWith()` returns a jQuery set containing only a paragraph.:

```
$("<div/>").replaceWith("<p></p>");
```

The `.replaceWith()` method can also take a function as its argument:

```
$('div.container').replaceWith(function() {
 return $(this).contents();
});
```

This results in being replaced by its three child s. The return value of the function may be an HTML string, DOM element, or jQuery object.

Example

On click, replace the button with a div containing the same word.

```
$("button").click(function () {
 $(this).replaceWith( "<div>" + $(this).text() + "</div>" );
});
```

Example

Replace all paragraphs with bold words.

```
$("p").replaceWith( "<b>Paragraph. </b>" );
```

Example

On click, replace each paragraph with a div that is already in the DOM and selected with the `$()` function. Notice it doesn't clone the object but rather moves it to replace the paragraph.

```
$("p").click(function () {
 $(this).replaceWith( $("div") );
});
```

Example
On button click, replace the containing div with its child divs and append the class name of the selected element to the paragraph.

```
$('button').bind("click", function() {
 var $container = $("div.container").replaceWith(function() {
  return $(this).contents();
 });

 $("p").append( $container.attr("class") );
});
```

General Attributes

removeProp(propertyName)
Remove a property for the set of matched elements.

Arguments
propertyName - The name of the property to set.

The `.removeProp()` method removes properties set by the `.prop()` method.

With some built-in properties of a DOM element or `window` object, browsers may generate an error if an attempt is made to remove the property. jQuery first assigns the value `undefined` to the property and ignores any error the browser generates. In general, it is only necessary to remove custom properties that have been set on an object, and not built-in (native) properties.

Note: Do not use this method to remove native properties such as checked, disabled, or selected. This will remove the property completely and, once removed, cannot be added again to element. Use `.prop()` to set these properties to `false` instead.

Example
Set a numeric property on a paragraph and then remove it.

```
var $para = $("p");
$para.prop("luggageCode", 1234);
$para.append("The secret luggage code is: ", String($para.prop("luggageCode")), ". ");
$para.removeProp("luggageCode");
$para.append("Now the secret luggage code is: ", String($para.prop("luggageCode")), ". ");
```

prop(propertyName)
Get the value of a property for the first element in the set of matched elements.

Arguments
propertyName - The name of the property to get.

The `.prop()` method gets the property value for only the *first* element in the matched set. It returns `undefined` for the value of a property that has not been set, or if the matched set has no elements. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

The difference between *attributes* and *properties* can be important in specific situations. **Before jQuery 1.6**, the `.attr()` method sometimes took property values into account when retrieving some attributes, which could cause inconsistent behavior. **As of jQuery 1.6**, the `.prop()` method provides a way to explicitly retrieve property values, while `.attr()` retrieves attributes.

For example, `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, and `defaultSelected` should be retrieved and set with the `.prop()` method. Prior to jQuery 1.6, these properties were retrievable with the `.attr()` method, but this was not within the scope of `attr`. These do not have corresponding attributes and are only properties.

Concerning boolean attributes, consider a DOM element defined by the HTML markup `<input type="checkbox" checked="checked" />`, and

assume it is in a JavaScript variable named `elem`:

`elem.checked`	`true`	(Boolean) Will change with checkbox state	
`$(elem).prop("checked")`	`true`	(Boolean) Will change with checkbox state	
`elem.getAttribute("checked")`	`"checked"`	(String) Initial state of the checkbox; does not change	
`$(elem).attr("checked")`	*(1.6)*	`"checked"`	(String) Initial s
`$(elem).attr("checked")`	*(1.6.1+)*	`"checked"`	(String) Will ch
`$(elem).attr("checked")`	*(pre-1.6)*	`true`	(Boolean) Cha

According to the W3C forms specification, the `checked` attribute is a *boolean attribute*, which means the corresponding property is true if the attribute is present at all-even if, for example, the attribute has no value or an empty string value. The preferred cross-browser-compatible way to determine if a checkbox is checked is to check for a "truthy" value on the element's property using one of the following:

- `if ( elem.checked )`
- `if ( $(elem).prop("checked") )`
- `if ( $(elem).is(":checked") )`

If using jQuery 1.6, the code `if ( $(elem).attr("checked") )` will retrieve the actual content *attribute*, which does not change as the checkbox is checked and unchecked. It is meant only to store the default or initial value of the checked property. To maintain backwards compatability, the `.attr()` method in jQuery 1.6.1+ will retrieve and update the property for you so no code for boolean attributes is required to be changed to `.prop()`. Nevertheless, the preferred way to retrieve a checked value is with one of the options listed above. To see how this works in the latest jQuery, check/uncheck the checkbox in the example below.

Example

Display the checked property and attribute of a checkbox as it changes.

```
$("input").change(function() {
 var $input = $(this);
 $("p").html(".attr('checked'): <b>" + $input.attr('checked') + "</b><br>"
       + ".prop('checked'): <b>" + $input.prop('checked') + "</b><br>"
       + ".is(':checked'): <b>" + $input.is(':checked') ) + "</b>";
}).change();
```

prop(propertyName, value)

Set one or more properties for the set of matched elements.

Arguments

propertyName - The name of the property to set.
value - A value to set for the property.

The `.prop()` method is a convenient way to set the value of properties-especially when setting multiple properties, using values returned by a function, or setting values on multiple elements at once. It should be used when setting `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, or `defaultSelected`. Since jQuery 1.6, these properties can no longer be set with the `.attr()` method. They do not have corresponding attributes and are only properties.

Properties generally affect the dynamic state of a DOM element without changing the serialized HTML attribute. Examples include the `value` property of input elements, the `disabled` property of inputs and buttons, or the `checked` property of a checkbox. The `.prop()` method should be used to set disabled and checked instead of the `.attr()` method. The `.val()` method should be used for getting and setting value.

```
$("input").prop("disabled", false);
$("input").prop("checked", true);
$("input").val("someValue");
```

Important: the `.removeProp()` method should not be used to set these properties to false. Once a native property is removed, it cannot be added again. See `.removeProp()` for more information.

Computed property values

By using a function to set properties, you can compute the value based on other properties of the element. For example, to toggle all checkboxes based off their individual values:

```
$("input[type='checkbox']").prop("checked", function( i, val ) {
 return !val;
});
```

Note: If nothing is returned in the setter function (ie. `function(index, prop){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example
Disable all checkboxes on the page.

```
$("input[type='checkbox']").prop({
 disabled: true
});
```

val()
Get the current value of the first element in the set of matched elements.

The `.val()` method is primarily used to get the values of form elements such as `input`, `select` and `textarea`. In the case of `<select multiple="multiple">` elements, the `.val()` method returns an array containing each selected option; if no option is selected, it returns `null`.

For selects and checkboxes, you can also use the :selected and :checked selectors to get at values, for example:

```
$('select.foo option:selected').val();  // get the value from a dropdown select
$('select.foo').val();          // get the value from a dropdown select even easier
$('input:checkbox:checked').val();    // get the value from a checked checkbox
$('input:radio[name=bar]:checked').val(); // get the value from a set of radio buttons
```

Note: At present, using `.val()` on textarea elements strips carriage return characters from the browser-reported value. When this value is sent to the server via XHR however, carriage returns are preserved (or added by browsers which do not include them in the raw value). A workaround for this issue can be achieved using a valHook as follows:

```
$.valHooks.textarea = {
    get: function( elem ) {
        return elem.value.replace( /r?n/g, "rn" );
    }
};
```

Example
Get the single value from a single select and an array of values from a multiple select and display their values.

```
function displayVals() {
   var singleValues = $("#single").val();
   var multipleValues = $("#multiple").val() || [];
   $("p").html("<b>Single:</b> " +
         singleValues +
         " <b>Multiple:</b> " +
         multipleValues.join(", "));
  }

  $("select").change(displayVals);
  displayVals();
```

Example
Find the value of an input box.

```
$("input").keyup(function () {
```

```
   var value = $(this).val();
   $("p").text(value);
  }).keyup();
```

val(value)

Set the value of each element in the set of matched elements.

Arguments

value - A string of text or an array of strings corresponding to the value of each matched element to set as selected/checked.

This method is typically used to set the values of form fields.

Passing an array of element values allows matching `<input type="checkbox">`, `<input type="radio">` and `<option>`s inside of n `<select multiple="multiple">` to be selected. In the case of `<input type="radio">`s that are part of a radio group and `<select multiple="multiple">` the other elements will be deselected.

The `.val()` method allows us to set the value by passing in a function. As of jQuery 1.4, the function is passed two arguments, the current element's index and its current value:

```
$('input:text.items').val(function( index, value ) {
 return value + ' ' + this.className;
});
```

This example appends the string " items" to the text inputs' values.

Example

Set the value of an input box.

```
$("button").click(function () {
   var text = $(this).text();
   $("input").val(text);
  });
```

Example

Use the function argument to modify the value of an input box.

```
$('input').bind('blur', function() {
  $(this).val(function( i, val ) {
   return val.toUpperCase();
  });
 });
```

Example

Set a single select, a multiple select, checkboxes and a radio button .

```
$("#single").val("Single2");
  $("#multiple").val(["Multiple2", "Multiple3"]);
  $("input").val(["check1","check2", "radio1" ]);
```

removeAttr(attributeName)

Remove an attribute from each element in the set of matched elements.

Arguments

attributeName - An attribute to remove; as of version 1.7, it can be a space-separated list of attributes.

The `.removeAttr()` method uses the JavaScript `removeAttribute()` function, but it has the advantage of being able to be called directly on a jQuery object and it accounts for different attribute naming across browsers.

Note: Removing an inline `onclick` event handler using `.removeAttr()` doesn't achieve the desired effect in Internet Explorer 6, 7, or 8. To avoid potential problems, use `.prop()` instead:

```
$element.prop("onclick", null);
console.log("onclick property: ", $element[0].onclick);
```

Example
Clicking the button enables the input next to it.

```
(function() {
 var inputTitle = $("input").attr("title");
 $("button").click(function () {
  var input = $(this).next();

  if ( input.attr("title") == inputTitle ) {
   input.removeAttr("title")
  } else {
   input.attr("title", inputTitle);
  }

  $("#log").html( "input title is now " + input.attr("title") );
 });
})();
```

attr(attributeName)
Get the value of an attribute for the first element in the set of matched elements.

Arguments
attributeName - The name of the attribute to get.

The `.attr()` method gets the attribute value for only the *first* element in the matched set. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

As of jQuery 1.6, the `.attr()` method returns `undefined` for attributes that have not been set. In addition, `.attr()` should not be used on plain objects, arrays, the window, or the document. To retrieve and change DOM properties, use the .prop() method.

Using jQuery's `.attr()` method to get the value of an element's attribute has two main benefits:
- **Convenience**: It can be called directly on a jQuery object and chained to other jQuery methods.
- **Cross-browser consistency**: The values of some attributes are reported inconsistently across browsers, and even across versions of a single browser. The `.attr()` method reduces such inconsistencies.

Note: Attribute values are strings with the exception of a few attributes such as value and tabindex.

Example
Find the title attribute of the first in the page.

```
var title = $("em").attr("title");
 $("div").text(title);
```

attr(attributeName, value)
Set one or more attributes for the set of matched elements.

Arguments
attributeName - The name of the attribute to set.
value - A value to set for the attribute.

The `.attr()` method is a convenient way to set the value of attributes-especially when setting multiple attributes or using values returned by a function. Consider the following image:

```
<img id="greatphoto" src="brush-seller.jpg" alt="brush seller" />
```

Setting a simple attribute

To change the `alt` attribute, simply pass the name of the attribute and its new value to the `.attr()` method:

```
$('#greatphoto').attr('alt', 'Beijing Brush Seller');
```

Add an attribute the same way:

```
$('#greatphoto')
.attr('title', 'Photo by Kelly Clark');
```

Setting several attributes at once

To change the `alt` attribute and add the `title` attribute at the same time, pass both sets of names and values into the method at once using a map (JavaScript object literal). Each key-value pair in the map adds or modifies an attribute:

```
$('#greatphoto').attr({
 alt: 'Beijing Brush Seller',
 title: 'photo by Kelly Clark'
});
```

When setting multiple attributes, the quotes around attribute names are optional.

WARNING: When setting the 'class' attribute, you must always use quotes!

Note: jQuery prohibits changing the `type` attribute on an `<input>` or `<button>` element and will throw an error in all browsers. This is because the `type` attribute cannot be changed in Internet Explorer.

Computed attribute values

By using a function to set attributes, you can compute the value based on other properties of the element. For example, to concatenate a new value with an existing value:

```
$('#greatphoto').attr('title', function(i, val) {
 return val + ' - photo by Kelly Clark'
});
```

This use of a function to compute attribute values can be particularly useful when modifying the attributes of multiple elements at once.

Note: If nothing is returned in the setter function (ie. `function(index, attr){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example

Set some attributes for all <img>s in the page.

```
$("img").attr({
 src: "/images/hat.gif",
 title: "jQuery",
 alt: "jQuery Logo"
});
$("div").text($("img").attr("alt"));
```

Example

Set the id for divs based on the position in the page.

```
$("div").attr("id", function (arr) {
 return "div-id" + arr;
})
.each(function () {
 $("span", this).html("(ID = '<b>" + this.id + "</b>')");
});
```

Example

Set the src attribute from title attribute on the image.

```
$("img").attr("src", function() {
  return "/images/" + this.title;
});
```

Style Properties

outerWidth([includeMargin])

Get the current computed width for the first element in the set of matched elements, including padding and border.

Arguments

includeMargin - A Boolean indicating whether to include the element's margin in the calculation.

Returns the width of the element, along with left and right padding, border, and optionally margin, in pixels.

If `includeMargin` is omitted or `false`, the padding and border are included in the calculation; if `true`, the margin is also included.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.

Example

Get the outerWidth of a paragraph.

```
var p = $("p:first");
$("p:last").text( "outerWidth:" + p.outerWidth()+ " , outerWidth(true):" + p.outerWidth(true) );
```

outerHeight([includeMargin])

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

Arguments

includeMargin - A Boolean indicating whether to include the element's margin in the calculation.

The top and bottom padding and border are always included in the `.outerHeight()` calculation; if the `includeMargin` argument is set to `true`, the margin (top and bottom) is also included.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.

Example

Get the outerHeight of a paragraph.

```
var p = $("p:first");
$("p:last").text( "outerHeight:" + p.outerHeight() + " , outerHeight(true):" + p.outerHeight(true) );
```

innerWidth()

Get the current computed width for the first element in the set of matched elements, including padding but not border.

This method returns the width of the element, including left and right padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.

Example

Get the innerWidth of a paragraph.

```
var p = $("p:first");
$("p:last").text( "innerWidth:" + p.innerWidth() );
```

innerHeight()

Get the current computed height for the first element in the set of matched elements, including padding but not border.

This method returns the height of the element, including top and bottom padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.

Example

Get the innerHeight of a paragraph.

```
var p = $("p:first");
$("p:last").text( "innerHeight:" + p.innerHeight() );
```

width()

Get the current computed width for the first element in the set of matched elements.

The difference between `.css(width)` and `.width()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.width()` method is recommended when an element's width needs to be used in a mathematical calculation.

This method is also able to find the width of the window and document.

```
$(window).width();  // returns width of browser viewport
$(document).width(); // returns width of HTML document
```

Note that `.width()` will always return the content width, regardless of the value of the CSS `box-sizing` property.

Example

Show various widths. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showWidth(ele, w) {
 $("div").text("The width for the " + ele +
        " is " + w + "px.");
}
$("#getp").click(function () {
 showWidth("paragraph", $("p").width());
});
$("#getd").click(function () {
 showWidth("document", $(document).width());
});
$("#getw").click(function () {
 showWidth("window", $(window).width());
});
```

width(value)

Set the CSS width of each element in the set of matched elements.

Arguments

value - An integer representing the number of pixels, or an integer along with an optional unit of measure appended (as a string).

When calling `.width("value")`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, any valid CSS measurement may be used for the width (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS width property does not include padding, border, or margin, unless the `box-sizing` CSS property is used.

If no explicit unit is specified (like "em" or "%") then "px" is assumed.

Note that `.width("value")` sets the width of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the `outerWidth` of the box instead of the content width.

Example
Change the width of each div the first time it is clicked (and change its color).

```
(function() {
 var modWidth = 50;
 $("div").one('click', function () {
  $(this).width(modWidth).addClass("mod");
 modWidth -= 8;
 });
})();
```

height()
Get the current computed height for the first element in the set of matched elements.

The difference between `.css('height')` and `.height()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.height()` method is recommended when an element's height needs to be used in a mathematical calculation.

This method is also able to find the height of the window and document.

```
$(window).height();  // returns height of browser viewport
$(document).height(); // returns height of HTML document
```

Note that `.height()` will always return the content height, regardless of the value of the CSS `box-sizing` property.

Note: Although `style` and `script` tags will report a value for `.width()` or `height()` when absolutely positioned and given `display:block`, it is strongly discouraged to call those methods on these tags. In addition to being a bad practice, the results may also prove unreliable.

Example
Show various heights. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showHeight(ele, h) {
   $("div").text("The height for the " + ele +
          " is " + h + "px.");
  }
  $("#getp").click(function () {
   showHeight("paragraph", $("p").height());
  });
  $("#getd").click(function () {
   showHeight("document", $(document).height());
  });
  $("#getw").click(function () {
   showHeight("window", $(window).height());
  });
```

height(value)
Set the CSS height of every matched element.

Arguments

value - An integer representing the number of pixels, or an integer with an optional unit of measure appended (as a string).

When calling `.height(value)`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, a valid CSS measurement must be provided for the height (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS height property does not include padding, border, or margin.

If no explicit unit was specified (like 'em' or '%') then "px" is concatenated to the value.

Note that `.height(value)` sets the height of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the outerHeight of the box instead of the content height.

Example
To set the height of each div on click to 30px plus a color change.

```
$("div").one('click', function () {
   $(this).height(30)
       .css({cursor:"auto", backgroundColor:"green"});
  });
```

scrollLeft()
Get the current horizontal position of the scroll bar for the first element in the set of matched elements.

The horizontal scroll position is the same as the number of pixels that are hidden from view to the left of the scrollable area. If the scroll bar is at the very left, or if the element is not scrollable, this number will be `0`.

Note: `.scrollLeft()`, when called directly or animated as a property using `.animate()`, will not work if the element it is being applied to is hidden.

Example
Get the scrollLeft of a paragraph.

```
var p = $("p:first");
$("p:last").text( "scrollLeft:" + p.scrollLeft() );
```

scrollLeft(value)
Set the current horizontal position of the scroll bar for each of the set of matched elements.

Arguments
value - An integer indicating the new position to set the scroll bar to.

The horizontal scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollLeft` positions the horizontal scroll of each matched element.

Example
Set the scrollLeft of a div.

```
$("div.demo").scrollLeft(300);
```

scrollTop()
Get the current vertical position of the scroll bar for the first element in the set of matched elements.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. If the scroll bar is at the very top, or if the element is not scrollable, this number will be `0`.

Example
Get the scrollTop of a paragraph.

```
var p = $("p:first");
$("p:last").text( "scrollTop:" + p.scrollTop() );
```

scrollTop(value)
Set the current vertical position of the scroll bar for each of the set of matched elements.

Arguments
value - An integer indicating the new position to set the scroll bar to.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollTop` positions the vertical scroll of each matched element.

Example
Set the scrollTop of a div.

```
$("div.demo").scrollTop(300);
```

position()
Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

The `.position()` method allows us to retrieve the current position of an element *relative to the offset parent*. Contrast this with `.offset()`, which retrieves the current position *relative to the document*. When positioning a new element near another one and within the same containing DOM element, `.position()` is the more useful.

Returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the position coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.

Example
Access the position of the second paragraph:

```
var p = $("p:first");
var position = p.position();
$("p:last").text( "left: " + position.left + ", top: " + position.top );
```

offset()
Get the current coordinates of the first element in the set of matched elements, relative to the document.

The `.offset()` method allows us to retrieve the current position of an element *relative to the document*. Contrast this with `.position()`, which retrieves the current position *relative to the offset parent*. When positioning a new element on top of an existing one for global manipulation (in particular, for implementing drag-and-drop), `.offset()` is the more useful.

`.offset()` returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the offset coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.

While it is possible to get the coordinates of elements with `visibility:hidden` set, `display:none` is excluded from the rendering tree and thus has a position that is undefined.

Example
Access the offset of the second paragraph:

```
var p = $("p:last");
var offset = p.offset();
p.html( "left: " + offset.left + ", top: " + offset.top );
```

Example
Click to see the offset.

```
$("*", document.body).click(function (e) {
```

```
 var offset = $(this).offset();
 e.stopPropagation();
 $("#result").text(this.tagName + " coords ( " + offset.left + ", " +
                 offset.top + " )");
});
```

offset(coordinates)

Set the current coordinates of every element in the set of matched elements, relative to the document.

Arguments

coordinates - An object containing the properties `top` and `left`, which are integers indicating the new top and left coordinates for the elements.

The `.offset()` setter method allows us to reposition an element. The element's position is specified *relative to the document*. If the element's `position` style property is currently `static`, it will be set to `relative` to allow for this repositioning.

Example

Set the offset of the second paragraph:

```
$("p:last").offset({ top: 10, left: 30 });
```

css(propertyName)

Get the value of a style property for the first element in the set of matched elements.

Arguments

propertyName - A CSS property.

The `.css()` method is a convenient way to get a style property from the first matched element, especially in light of the different ways browsers access most of those properties (the `getComputedStyle()` method in standards-based browsers versus the `currentStyle` and `runtimeStyle` properties in Internet Explorer) and the different terms browsers use for certain properties. For example, Internet Explorer's DOM implementation refers to the `float` property as `styleFloat`, while W3C standards-compliant browsers refer to it as `cssFloat`. The `.css()` method accounts for such differences, producing the same result no matter which term we use. For example, an element that is floated left will return the string `left` for each of the following three lines:

- `$('div.left').css('float');`
- `$('div.left').css('cssFloat');`
- `$('div.left').css('styleFloat');`

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css('background-color')` and `.css('backgroundColor')`. Different browsers may return CSS color values that are logically but not textually equal, e.g., #FFF, #ffffff, and rgb(255,255,255).

Shorthand CSS properties (e.g. margin, background, border) are not supported. For example, if you want to retrieve the rendered margin, use: `$(elem).css('marginTop')` and `$(elem).css('marginRight')`, and so on.

Example

To access the background color of a clicked div.

```
$("div").click(function () {
 var color = $(this).css("background-color");
 $("#result").html("That div is <span style='color:" +
           color + ";'>" + color + "</span>.");
});
```

css(propertyName, value)

Set one or more CSS properties for the set of matched elements.

Arguments

propertyName - A CSS property name.
value - A value to set for the property.

As with the `.prop()` method, the `.css()` method makes setting properties of elements quick and easy. This method can take either a property name

and value as separate parameters, or a single map of key-value pairs (JavaScript object notation).

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css({'background-color': '#ffe', 'border-left': '5px solid #ccc'})` and `.css({backgroundColor: '#ffe', borderLeft: '5px solid #ccc'})`. Notice that with the DOM notation, quotation marks around the property names are optional, but with CSS notation they're required due to the hyphen in the name.

When using `.css()` as a setter, jQuery modifies the element's `style` property. For example, `$('#mydiv').css('color', 'green')` is equivalent to `document.getElementById('mydiv').style.color = 'green'`. Setting the value of a style property to an empty string - e.g. `$('#mydiv').css('color', '')` - removes that property from an element if it has already been directly applied, whether in the HTML style attribute, through jQuery's `.css()` method, or through direct DOM manipulation of the `style` property. It does not, however, remove a style that has been applied with a CSS rule in a stylesheet or `<style>` element.

As of jQuery 1.6, `.css()` accepts relative values similar to `.animate()`. Relative values are a string starting with += or -= to increment or decrement the current value. For example, if an element's padding-left was 10px, `.css( "padding-left", "+=15" )` would result in a total padding-left of 25px.

As of jQuery 1.4, `.css()` allows us to pass a function as the property value:

```
$('div.example').css('width', function(index) {
 return index * 50;
});
```

This example sets the widths of the matched elements to incrementally larger values.

Note: If nothing is returned in the setter function (ie. `function(index, style){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example
To change the color of any paragraph to red on mouseover event.

```
$("p").mouseover(function () {
  $(this).css("color","red");
 });
```

Example
Increase the width of #box by 200 pixels

```
$("#box").one( "click", function () {
  $( this ).css( "width","+=200" );
 });
```

Example
To highlight a clicked word in the paragraph.

```
var words = $("p:first").text().split(" ");
 var text = words.join("</span> <span>");
 $("p:first").html("<span>" + text + "</span>");
 $("span").click(function () {
  $(this).css("background-color","yellow");
 });
```

Example
To set the color of all paragraphs to red and background to blue:

```
$("p").hover(function () {
  $(this).css({'background-color' : 'yellow', 'font-weight' : 'bolder'});
 }, function () {
  var cssObj = {
   'background-color' : '#ddd',
   'font-weight' : '',
   'color' : 'rgb(0,40,244)'
  }
```

```
  $(this).css(cssObj);
 });
```

Example

Increase the size of a div when you click it:

```
$("div").click(function() {
  $(this).css({
   width: function(index, value) {
    return parseFloat(value) * 1.2;
   },
   height: function(index, value) {
    return parseFloat(value) * 1.2;
   }

  });
 });
```

Miscellaneous

Collection Manipulation

each(function(index, Element))
Iterate over a jQuery object, executing a function for each matched element.

Arguments
function(index, Element) - A function to execute for each matched element.

The `.each()` method is designed to make DOM looping constructs concise and less error-prone. When called it iterates over the DOM elements that are part of the jQuery 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 keyword `this` refers to the element.

Suppose we had a simple unordered list on the page:

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

We can select the list items and iterate across them:

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

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

0: foo1: bar

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

Example
Iterates over three divs and sets their color property.

```
$(document.body).click(function () {
   $("div").each(function (i) {
    if (this.style.color != "blue") {
     this.style.color = "blue";
    } else {
     this.style.color = "";
    }
   });
  });
```

Example
If you want to have the jQuery object instead of the regular DOM element, use the $(this) function, for example:

```
$("span").click(function () {
   $("li").each(function(){
    $(this).toggleClass("example");
   });
  });
```

Example
You can use 'return' to break out of each() loops early.

```
$("button").click(function () {
```

```
  $("div").each(function (index, domEle) {
   // domEle == this
   $(domEle).css("backgroundColor", "yellow");
   if ($(this).is("#stop")) {
    $("span").text("Stopped at div index #" + index);
    return false;
   }
  });
 });
```

jQuery.param(obj)

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

Arguments

obj - An array or object to serialize.

This function is used internally to convert form element values into a serialized string representation (See .serialize() for more information).

As of jQuery 1.3, the return value of a function is used instead of the function as a String.

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. You can disable this functionality globally by setting `jQuery.ajaxSettings.traditional = true;`.

If the object passed is in an Array, it must be an array of objects in the format returned by .serializeArray()

```
[{name:"first",value:"Rick"},
{name:"last",value:"Astley"},
{name:"job",value:"Rock Star"}]
```

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

Note: Because there is no universally agreed-upon specification for param strings, it is not possible to encode complex data structures using this method in a manner that works ideally across all languages supporting such input. Until such time that there is, the `$.param` method will remain in its current form.

In jQuery 1.4, HTML5 input elements are also serialized.

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));

alert(recursiveEncoded);
alert(recursiveDecoded);
```

The values of `recursiveEncoded` and `recursiveDecoded` are alerted 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);

alert(shallowEncoded);
alert(shallowDecoded);
```

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

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

Example
Serialize a key/value object.

```
var params = { width:1680, height:1050 };
  var str = jQuery.param(params);
  $("#results").text(str);
```

Example
Serialize a few complex objects

```
// <=1.3.2:
$.param({ a: [2,3,4] }) // "a=2&a=3&a=4"
// >=1.4:
$.param({ a: [2,3,4] }) // "a[]=2&a[]=3&a[]=4"

// <=1.3.2:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a=[object+Object]&d=3&d=4&d=[object+Object]"
// >=1.4:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a[b]=1&a[c]=2&d[]=3&d[]=4&d[2][e]=5"
```

Data Storage

removeData([name])
Remove a previously-stored piece of data.

Arguments
name - A string naming the piece of data to delete.

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. Removing data from jQuery's internal `.data()` cache does not effect any HTML5 `data-` attributes in a document; use `.removeAttr()` to remove those.

When using `.removeData("name")`, jQuery will attempt to locate a `data-` attribute on the element if no property by that name is in the internal data cache. To avoid a re-query of the `data-` attribute, set the name to a value of either `null` or `undefined` (e.g. `.data("name", undefined)`) rather than using `.removeData()`.

As of jQuery 1.7, when called with an array of keys or a string of space-separated keys, `.removeData()` deletes the value of each key in that array or string.

As of jQuery 1.4.3, calling `.removeData()` will cause the value of the property being removed to revert to the value of the data attribute of the same name in the DOM, rather than being set to `undefined`.

Example
Set a data store for 2 names then remove one of them.

```
$("span:eq(0)").text("" + $("div").data("test1"));
  $("div").data("test1", "VALUE-1");
  $("div").data("test2", "VALUE-2");
  $("span:eq(1)").text("" + $("div").data("test1"));
  $("div").removeData("test1");
  $("span:eq(2)").text("" + $("div").data("test1"));
  $("span:eq(3)").text("" + $("div").data("test2"));
```

data(key, value)
Store arbitrary data associated with the matched elements.

Arguments
key - A string naming the piece of data to set.
value - The new data value; it can be any Javascript type including Array or Object.

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 later:

```
$('body').data('foo', 52);
$('body').data('bar', { myType: 'test', count: 40 });

$('body').data('foo'); // 52
$('body').data(); // {foo: 52, bar: { myType: 'test', count: 40 }}
```

In jQuery 1.4.3 setting an element's data object with `.data(obj)` extends the data previously stored with that element. jQuery itself uses the `.data()` method to save information under the names 'events' and 'handle', and also reserves any data name starting with an underscore ('_') for internal use.

Prior to jQuery 1.4.3 (starting in jQuery 1.4) the .data() method completely replaced all data, instead of just extending the data object. If you are using third-party plugins it may not be advisable to completely replace the element's data object, since plugins may have also set data.

Due to the way browsers interact with plugins and external code, the `.data()` method cannot be used on `<object>` (unless it's a Flash plugin), `<applet>` or `<embed>` elements.

Example
Store then retrieve a value from the div element.

```
$("div").data("test", { first: 16, last: "pizza!" });
$("span:first").text($("div").data("test").first);
$("span:last").text($("div").data("test").last);
```

data(key)
Returns value at named data store for the first element in the jQuery collection, as set by data(name, value).

Arguments
key - Name of the data stored.

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 retrieve several distinct values for a single element one at a time, or as a set:

```
alert($('body').data('foo'));
alert($('body').data());
```

The above lines alert the data values that were set on the `body` element. If no data at all was set on that element, `undefined` is returned.

```
alert( $("body").data("foo")); //undefined
$("body").data("bar", "foobar");
alert( $("body").data("bar")); //foobar
```

HTML5 data-* Attributes

As of jQuery 1.4.3 HTML 5 data- attributes will be automatically pulled in to jQuery's data object. The treatment of attributes with embedded dashes was changed in jQuery 1.6 to conform to the W3C HTML5 specification.

For example, given the following HTML:

```
<div data-role="page" data-last-value="43" data-hidden="true" data-options='{"name":"John"}'></div>
```

All of the following jQuery code will work.

```
$("div").data("role") === "page";
$("div").data("lastValue") === 43;
$("div").data("hidden") === true;
$("div").data("options").name === "John";
```

Every attempt is made to convert the string to a JavaScript value (this includes booleans, numbers, objects, arrays, and null) otherwise it is left as a string. To retrieve the value's attribute as a string without any attempt to convert it, use the `attr()` method. When the data attribute is an object (starts with '{') or array (starts with '[') then `jQuery.parseJSON` is used to parse the string; it must follow valid JSON syntax*including quoted property names*. The data- attributes are pulled in the first time the data property is accessed and then are no longer accessed or mutated (all data values are then stored internally in jQuery).

Calling `.data()` with no parameters retrieves all of the values as a JavaScript object. This object can be safely cached in a variable as long as a new object is not set with `.data(obj)`. Using the object directly to get or set values is faster than making individual calls to `.data()` to get or set each value:

```
var mydata = $("#mydiv").data();
if ( mydata.count < 9 ) {
  mydata.count = 43;
  mydata.status = "embiggened";
}
```

Example

Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value;

 switch ($("button").index(this)) {
  case 0 :
   value = $("div").data("blah");
   break;
  case 1 :
   $("div").data("blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   $("div").data("blah", 86);
   value = "Stored!";
   break;
  case 3 :
   $("div").removeData("blah");
```

```
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

DOM Element Methods

toArray()

Retrieve all the DOM elements contained in the jQuery set, as an array.

`.toArray()` returns all of the elements in the jQuery set:

```
alert($('li').toArray());
```

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

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

Example

Selects all divs in the document and returns the DOM Elements as an Array, then uses the built-in reverse-method to reverse that array.

```
function disp(divs) {
   var a = [];
   for (var i = 0; i < divs.length; i++) {
    a.push(divs[i].innerHTML);
   }
   $("span").text(a.join(" "));
  }

  disp( $("div").toArray().reverse() );
```

index()

Search for a given element from among the matched elements.

Return Values

If no argument is passed to the `.index()` method, the return value is an integer indicating the position of the first element within the jQuery object relative to its sibling elements.

If `.index()` is called on a collection of elements and a DOM element or jQuery object is passed in, `.index()` returns an integer indicating the position of the passed element relative to the original collection.

If a selector string is passed as an argument, `.index()` returns an integer indicating the position of the original element relative to the elements matched by the selector. If the element is not found, `.index()` will return -1.

Detail

The complementary operation to `.get()`, which accepts an index and returns a DOM node, `.index()` can take a DOM node and returns an index. Suppose we have 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 retrieve 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');
alert('Index: ' + $('li').index(listItem));
We get back the zero-based position of the list item:
```

Index: 1

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

```
var listItem = $('#bar');
alert('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)');
alert('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 which also matches this selector is located.

```
var listItem = $('#bar');
alert('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:

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

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

Index: 1

Example

On click, returns the index (based zero) of that div in the page.

```
$("div").click(function () {
 // this is the dom element clicked
```

```
 var index = $("div").index(this);
 $("span").text("That was div index #" + index);
});
```

Example
Returns the index for the element with ID bar.

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

Example
Returns the index for the first item in the jQuery collection.

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

Example
Returns the index for the element with ID bar in relation to all <li> elements.

```
$('div').html('Index: ' + $('#bar').index('li') );
```

Example
Returns the index for the element with ID bar in relation to its siblings.

```
var barIndex = $('#bar').index();
$('div').html( 'Index: ' + barIndex );
```

Example
Returns -1, as there is no element with ID foobar.

```
var foobar = $("li").index( $('#foobar') );
$('div').html('Index: ' + foobar);
```

get([index])
Retrieve the DOM elements matched by the jQuery object.

Arguments
index - A zero-based integer indicating which element to retrieve.

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:

```
alert($('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:

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

Since 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:

```
alert($('li')[0]);
```

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

```
alert($('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">

Example
Selects all divs in the document and returns the DOM Elements as an Array, then uses the built-in reverse-method to reverse that array.

```
function disp(divs) {
   var a = [];
   for (var i = 0; i < divs.length; i++) {
    a.push(divs[i].innerHTML);
   }
   $("span").text(a.join(" "));
  }

  disp( $("div").get().reverse() );
```

Example
Gives the tag name of the element clicked on.

```
$("*", document.body).click(function (e) {
   e.stopPropagation();
   var domEl = $(this).get(0);
   $("span:first").text("Clicked on - " + domEl.tagName);
  });
```

size()
Return the number of elements in the jQuery object.

The `.size()` method is functionally equivalent to the `.length` property; however, **the** `.length` property is preferred because it does not have the overhead of a function call.

Given a simple unordered list on the page:

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

Both `.size()` and `.length` identify the number of items:

```
alert( "Size: " + $("li").size() );
alert( "Size: " + $("li").length );
```

This results in two alerts:

Size: 2

Size: 2

Example
Count the divs. Click to add more.

```
$(document.body)
.click(function() {
 $(this).append( $("<div>") );
 var n = $("div").size();
 $("span").text("There are " + n + " divs. Click to add more.");
})
// trigger the click to start
.click();
```

Setup Methods

jQuery.noConflict([removeAll])
Relinquish jQuery's control of the $ variable.

Arguments
removeAll - A Boolean indicating whether to remove all jQuery variables from the global scope (including jQuery itself).

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 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.

Example
Maps the original object that was referenced by $ back to $.

```
jQuery.noConflict();
// Do something with jQuery
jQuery("div p").hide();
// Do something with another library's $()
$("content").style.display = 'none';
```

Example
Reverts the $ alias and then creates and executes a function to provide the $ as a jQuery alias inside the functions scope. Inside the function the original $ object is not available. This works well for most plugins that don't rely on any other library.

```
jQuery.noConflict();
(function($) {
 $(function() {
  // more code using $ as alias to jQuery
 });
})(jQuery);
// other code using $ as an alias to the other library
```

Example
You can chain the jQuery.noConflict() with the shorthand ready for a compact code.

```
jQuery.noConflict()(function(){
  // code using jQuery
});
// other code using $ as an alias to the other library
```

Example
Creates a different alias instead of jQuery to use in the rest of the script.

```
var j = jQuery.noConflict();
// Do something with jQuery
j("div p").hide();
// Do something with another library's $()
$("content").style.display = 'none';
```

Example
Completely move jQuery to a new namespace in another object.

```
var dom = {};
dom.query = jQuery.noConflict(true);
```

Offset

offsetParent()

Get the closest ancestor element that is positioned.

Given a jQuery object that represents a set of DOM elements, the `.offsetParent()` method allows us to search through the ancestors of these elements in the DOM tree and construct a new jQuery object wrapped around the closest positioned ancestor. An element is said to be positioned if it has a CSS position attribute of `relative`, `absolute`, or `fixed`. This information is useful for calculating offsets for performing animations and placing objects on the page.

Consider a page with a basic nested list on it, with a positioned element:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii" style="position: relative;">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at item A, we can find its positioned ancestor:

```
$('li.item-a').offsetParent().css('background-color', 'red');
```

This will change the color of list item II, which is positioned.

Example

Find the offsetParent of item "A."

```
$('li.item-a').offsetParent().css('background-color', 'red');
```

scrollLeft()

Get the current horizontal position of the scroll bar for the first element in the set of matched elements.

The horizontal scroll position is the same as the number of pixels that are hidden from view to the left of the scrollable area. If the scroll bar is at the very left, or if the element is not scrollable, this number will be `0`.

Note: `.scrollLeft()`, when called directly or animated as a property using `.animate()`, will not work if the element it is being applied to is hidden.

Example

Get the scrollLeft of a paragraph.

```
var p = $("p:first");
$("p:last").text( "scrollLeft:" + p.scrollLeft() );
```

scrollLeft(value)
Set the current horizontal position of the scroll bar for each of the set of matched elements.

Arguments
value - An integer indicating the new position to set the scroll bar to.

The horizontal scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollLeft` positions the horizontal scroll of each matched element.

Example
Set the scrollLeft of a div.

```
$("div.demo").scrollLeft(300);
```

scrollTop()
Get the current vertical position of the scroll bar for the first element in the set of matched elements.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. If the scroll bar is at the very top, or if the element is not scrollable, this number will be `0`.

Example
Get the scrollTop of a paragraph.

```
var p = $("p:first");
$("p:last").text( "scrollTop:" + p.scrollTop() );
```

scrollTop(value)
Set the current vertical position of the scroll bar for each of the set of matched elements.

Arguments
value - An integer indicating the new position to set the scroll bar to.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollTop` positions the vertical scroll of each matched element.

Example
Set the scrollTop of a div.

```
$("div.demo").scrollTop(300);
```

position()
Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

The `.position()` method allows us to retrieve the current position of an element *relative to the offset parent*. Contrast this with `.offset()`, which retrieves the current position *relative to the document*. When positioning a new element near another one and within the same containing DOM element, `.position()` is the more useful.

Returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the position coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.

Example
Access the position of the second paragraph:

```
var p = $("p:first");
var position = p.position();
$("p:last").text( "left: " + position.left + ", top: " + position.top );
```

offset()

Get the current coordinates of the first element in the set of matched elements, relative to the document.

The `.offset()` method allows us to retrieve the current position of an element *relative to the document*. Contrast this with `.position()`, which retrieves the current position *relative to the offset parent*. When positioning a new element on top of an existing one for global manipulation (in particular, for implementing drag-and-drop), `.offset()` is the more useful.

`.offset()` returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the offset coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.

While it is possible to get the coordinates of elements with `visibility:hidden` set, `display:none` is excluded from the rendering tree and thus has a position that is undefined.

Example

Access the offset of the second paragraph:

```
var p = $("p:last");
var offset = p.offset();
p.html( "left: " + offset.left + ", top: " + offset.top );
```

Example

Click to see the offset.

```
$("*", document.body).click(function (e) {
 var offset = $(this).offset();
 e.stopPropagation();
 $("#result").text(this.tagName + " coords ( " + offset.left + ", " +
                offset.top + " )");
});
```

offset(coordinates)

Set the current coordinates of every element in the set of matched elements, relative to the document.

Arguments

coordinates - An object containing the properties `top` and `left`, which are integers indicating the new top and left coordinates for the elements.

The `.offset()` setter method allows us to reposition an element. The element's position is specified *relative to the document*. If the element's `position` style property is currently `static`, it will be set to `relative` to allow for this repositioning.

Example

Set the offset of the second paragraph:

```
$("p:last").offset({ top: 10, left: 30 });
```

Plugin Authoring

Properties

Properties of jQuery Object Instances

Properties of the Global jQuery Object

jQuery.fx.interval
The rate (in milliseconds) at which animations fire.

This property can be manipulated to adjust the number of frames per second at which animations will run. The default is 13 milliseconds. Making this a lower number could make the animations run smoother in faster browsers (such as Chrome) but there may be performance and CPU implications of doing so.

Since jQuery uses one global interval, no animation should be running or all animations should stop for the change of this property to take effect.

Note: `jQuery.fx.interval` currently has no effect in browsers that support the `requestAnimationFrame` property, such as Google Chrome 11. This behavior is subject to change in a future release.

Example
Cause all animations to run with less frames.

```
jQuery.fx.interval = 100;

$("input").click(function(){
 $("div").toggle( 3000 );
});
```

jQuery.fx.off
Globally disable all animations.

When this property is set to `true`, all animation methods will immediately set elements to their final state when called, rather than displaying an effect. This may be desirable for a couple reasons:

- jQuery is being used on a low-resource device.
- Users are encountering accessibility problems with the animations (see the article Turn Off Animation for more information).

Animations can be turned back on by setting the property to `false`.

Example
Toggle animation on and off

```
var toggleFx = function() {
 $.fx.off = !$.fx.off;
};
toggleFx();

$("button").click(toggleFx)

$("input").click(function(){
 $("div").toggle("slow");
});
```

jQuery.browser
Contains flags for the useragent, read from navigator.userAgent. **We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.**

The `$.browser` property provides information about the web browser that is accessing the page, as reported by the browser itself. It contains flags for each of the four most prevalent browser classes (Internet Explorer, Mozilla, Webkit, and Opera) as well as version information.

Available flags are:

- webkit (as of jQuery 1.4)
- safari (deprecated)
- opera
- msie
- mozilla

This property is available immediately. It is therefore safe to use it to determine whether or not to call `$(document).ready()`.The `$.browser` property is deprecated in jQuery 1.3, and its functionality may be moved to a team-supported plugin in a future release of jQuery.

Because `$.browser` uses `navigator.userAgent` to determine the platform, it is vulnerable to spoofing by the user or misrepresentation by the browser itself. It is always best to avoid browser-specific code entirely where possible. The `$.support` property is available for detection of support for particular features rather than relying on `$.browser`.

Example

Show the browser info.

```
jQuery.each(jQuery.browser, function(i, val) {
   $("<div>" + i + " : <span>" + val + "</span>")
        .appendTo( document.body );
  });
```

Example

Returns true if the current useragent is some version of Microsoft's Internet Explorer.

```
$.browser.msie;
```

Example

Alerts "this is WebKit!" only for WebKit browsers

```
if ($.browser.webkit) {
  alert( "this is webkit!" );
 }
```

Example

Alerts "Do stuff for Firefox 3" only for Firefox 3 browsers.

```
var ua = $.browser;
 if ( ua.mozilla && ua.version.slice(0,3) == "1.9" ) {
  alert( "Do stuff for firefox 3" );
 }
```

Example

Set a CSS property that's specific to a particular browser.

```
if ( $.browser.msie ) {
  $("#div ul li").css( "display","inline" );
 } else {
  $("#div ul li").css( "display","inline-table" );
 }
```

jQuery.browser.version

The version number of the rendering engine for the user's browser.

Here are some typical results:

- Internet Explorer: 6.0, 7.0, 8.0
- Mozilla/Firefox/Flock/Camino: 1.7.12, 1.8.1.3, 1.9
- Opera: 10.06, 11.01

- Safari/Webkit: 312.8, 418.9

Note that IE8 claims to be 7 in Compatibility View.

Example
Returns the version number of the rendering engine used by the user's current browser. For example, FireFox 4 returns 2.0 (the version of the Gecko rendering engine it utilizes).

```
$("p").html( "The version number of the rendering engine your browser uses is: <span>" +
        $.browser.version + "</span>" );
```

Example
Alerts the version of IE's rendering engine that is being used:

```
if ( $.browser.msie ) {
 alert( $.browser.version );
}
```

Example
Often you only care about the "major number," the whole number, which you can get by using JavaScript's built-in `parseInt()` function:

```
if ( $.browser.msie ) {
 alert( parseInt($.browser.version, 10) );
}
```

jQuery.support
A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Rather than using `$.browser` to detect the current user agent and alter the page presentation based on which browser is running, it is a good practice to perform **feature detection**. This means that prior to executing code which relies on a browser feature, we test to ensure that the feature works properly. To make this process simpler, jQuery performs many such tests and makes the results available to us as properties of the `jQuery.support` object.

The values of all the support properties are determined using feature detection (and do not use any form of browser sniffing).

Following are a few resources that explain how feature detection works:
- http://peter.michaux.ca/articles/feature-detection-state-of-the-art-browser-scripting
- http://www.jibbering.com/faq/faq_notes/not_browser_detect.html
- http://yura.thinkweb2.com/cft/

While jQuery includes a number of properties, developers should feel free to add their own as their needs dictate. Many of the `jQuery.support` properties are rather low-level, so they are most useful for plugin and jQuery core development, rather than general day-to-day development. Since jQuery requires these tests internally, they must be performed on *every* page load; for that reason this list is kept short and limited to features needed by jQuery itself.

The tests included in `jQuery.support` are as follows:
- `ajax` is equal to true if a browser is able to create an `XMLHttpRequest` object.
- `boxModel` is equal to true if the page is rendering according to the W3C CSS Box Model (is currently false in IE 6 and 7 when they are in Quirks Mode). This property is null until document ready occurs.
- `changeBubbles` is equal to true if the change event bubbles up the DOM tree, as required by the W3C DOM event model. (It is currently false in IE, and jQuery simulates bubbling).
- `checkClone` is equal to true if a browser correctly clones the checked state of radio buttons or checkboxes in document fragments.
- `checkOn` is equal to true if the value of a checkbox defaults to "on" when no value is specified.
- `cors` is equal to true if a browser can create an `XMLHttpRequest` object and if that `XMLHttpRequest` object has a `withCredentials` property. To enable cross-domain requests in environments that do not support cors yet but do allow cross-domain XHR requests (windows gadget, etc), set `$.support.cors = true;`.CORS WD
- `cssFloat` is equal to true if the name of the property containing the CSS float value is .cssFloat, as defined in the CSS Spec. (It is currently false in IE, it uses styleFloat instead).
- `hrefNormalized` is equal to true if the `.getAttribute()` method retrieves the `href` attribute of elements unchanged, rather than normalizing it to a fully-qualified URL. (It is currently false in IE, the URLs are normalized). DOM l3 spec

• `htmlSerialize` is equal to true if the browser is able to serialize/insert `<link>` elements using the `.innerHTML` property of elements. (is currently false in IE). HTML5 WD

• `leadingWhitespace` is equal to true if the browser inserts content with .innerHTML exactly as provided-specifically, if leading whitespace characters are preserved. (It is currently false in IE 6-8). HTML5 WD

• `noCloneChecked` is equal to true if cloned DOM elements copy over the state of the `.checked` expando. (It is currently false in IE). (Added in jQuery 1.5.1)

• `noCloneEvent` is equal to true if cloned DOM elements are created without event handlers (that is, if the event handlers on the source element are not cloned). (It is currently false in IE). DOM l2 spec

• `opacity` is equal to true if a browser can properly interpret the opacity style property. (It is currently false in IE, it uses alpha filters instead). CSS3 spec

• `optDisabled` is equal to true if option elements within disabled select elements are not automatically marked as disabled. HTML5 WD

• `optSelected` is equal to true if an `<option>` element that is selected by default has a working `selected` property. HTML5 WD

• `scriptEval()` is equal to true if inline scripts are automatically evaluated and executed when inserted into the document using standard DOM manipulation methods such as `.appendChild()` and `.createTextNode()`. (It is currently false in IE, it uses `.text` to insert executable scripts). **Note: No longer supported; removed in jQuery 1.6. Prior to jQuery 1.5.1**, the `scriptEval()` method was the static `scriptEval` property. The change to a method allowed the test to be deferred until first use to prevent content security policy inline-script violations. HTML5 WD

• `style` is equal to true if inline styles for an element can be accessed through the DOM attribute called style, as required by the DOM Level 2 specification. In this case, `.getAttribute('style')` can retrieve this value; in Internet Explorer, `.cssText` is used for this purpose. DOM l2 Style spec

• `submitBubbles` is equal to true if the submit event bubbles up the DOM tree, as required by the W3C DOM event model. (It is currently false in IE, and jQuery simulates bubbling).

• `tbody` is equal to true if an empty `<table>` element can exist without a `<tbody>` element. According to the HTML specification, this sub-element is optional, so the property should be true in a fully-compliant browser. If false, we must account for the possibility of the browser injecting `<tbody>` tags implicitly. (It is currently false in IE, which automatically inserts `tbody` if it is not present in a string assigned to `innerHTML`). HTML5 spec

Example

Returns the box model for the iframe.

```
$("p").html("This frame uses the W3C box model: <span>" +
        jQuery.support.boxModel + "</span>");
```

Example

Returns false if the page is in QuirksMode in Internet Explorer

```
jQuery.support.boxModel
```

Selectors

Attribute

attributeContainsPrefix
Selects elements that have the specified attribute with a value either equal to a given string or starting with that string followed by a hyphen (-).

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector was introduced into the CSS specification to handle language attributes.

Example
Finds all links with an hreflang attribute that is english.

```
$('a[hreflang|="en"]').css('border','3px dotted green');
```

attributeContainsWord
Selects elements that have the specified attribute with a value containing a given word, delimited by spaces.

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector matches the test string against each word in the attribute value, where a "word" is defined as a string delimited by whitespace. The selector matches if the test string is exactly equal to any of the words.

Example
Finds all inputs with a name attribute that contains the word 'man' and sets the value with some text.

```
$('input[name~="man"]').val('mr. man is in it!');
```

attributeMultiple
Matches elements that match all of the specified attribute filters.

Arguments
attributeFilter1 - An attribute filter.
attributeFilter2 - Another attribute filter, reducing the selection even more
attributeFilterN - As many more attribute filters as necessary

Example
Finds all inputs that have an id attribute and whose name attribute ends with man and sets the value.

```
$('input[id][name$="man"]').val('only this one');
```

attributeContains
Selects elements that have the specified attribute with a value containing the a given substring.

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This is the most generous of the jQuery attribute selectors that match against a value. It will select an element if the selector's string appears anywhere within the element's attribute value. Compare this selector with the Attribute Contains Word selector (e.g. [attr~="word"]), which is more appropriate in many cases.

Example

Finds all inputs with a name attribute that contains 'man' and sets the value with some text.

```
$('input[name*="man"]').val('has man in it!');
```

attributeEndsWith

Selects elements that have the specified attribute with a value ending exactly with a given string. The comparison is case sensitive.

Arguments

attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

Example

Finds all inputs with an attribute name that ends with 'letter' and puts text in them.

```
$('input[name$="letter"]').val('a letter');
```

attributeStartsWith

Selects elements that have the specified attribute with a value beginning exactly with a given string.

Arguments

attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector can be useful for identifying elements in pages produced by server-side frameworks that produce HTML with systematic element IDs. However it will be slower than using a class selector so leverage classes, if you can, to group like elements.

Example

Finds all inputs with an attribute name that starts with 'news' and puts text in them.

```
$('input[name^="news"]').val('news here!');
```

attributeNotEqual

Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

Arguments

attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector is equivalent to `:not([attr="value"])`.

Example

Finds all inputs that don't have the name 'newsletter' and appends text to the span next to it.

```
$('input[name!="newsletter"]').next().append('<b>; not newsletter</b>');
```

attributeEquals

Selects elements that have the specified attribute with a value exactly equal to a certain value.

Arguments

attribute - An attribute name.
value - An attribute value. **Can be either an unquoted single word or a quoted string.**

Example

Finds all inputs with a value of "Hot Fuzz" and changes the text of the next sibling span.

```
$('input[value="Hot Fuzz"]').next().text(" Hot Fuzz");
```

attributeHas

Selects elements that have the specified attribute, with any value.

Arguments
attribute - An attribute name.

Example
Bind a single click that adds the div id to its text.

```
$('div[id]').one('click', function(){
   var idString = $(this).text() + ' = ' + $(this).attr('id');
   $(this).text(idString);
  });
```

Basic

multiple
Selects the combined results of all the specified selectors.

Arguments
selector1 - Any valid selector.
selector2 - Another valid selector.
selectorN - As many more valid selectors as you like.

You can specify any number of selectors to combine into a single result. This multiple expression combinator is an efficient way to select disparate elements. The order of the DOM elements in the returned jQuery object may not be identical, as they will be in document order. An alternative to this combinator is the .add() method.

Example
Finds the elements that match any of these three selectors.

```
$("div,span,p.myClass").css("border","3px solid red");
```

Example
Show the order in the jQuery object.

```
var list = $("div,p,span").map(function () {
   return this.tagName;
  }).get().join(", ");
  $("b").append(document.createTextNode(list));
```

all
Selects all elements.

Caution: The all, or universal, selector is extremely slow, except when used by itself.

Example
Finds every element (including head, body, etc) in the document.

```
var elementCount = $("*").css("border","3px solid red").length;
$("body").prepend("<h3>" + elementCount + " elements found</h3>");
```

Example
A common way to select all elements is to find within document.body so elements like head, script, etc are left out.

```
var elementCount = $("#test").find("*").css("border","3px solid red").length;
$("body").prepend("<h3>" + elementCount + " elements found</h3>");
```

class

Selects all elements with the given class.

Arguments
class - A class to search for. An element can have multiple classes; only one of them must match.

For class selectors, jQuery uses JavaScript's native `getElementsByClassName()` function if the browser supports it.

Example
Finds the element with the class "myClass".

```
$(".myClass").css("border","3px solid red");
```

Example
Finds the element with both "myclass" and "otherclass" classes.

```
$(".myclass.otherclass").css("border","13px solid red");
```

element
Selects all elements with the given tag name.

Arguments
element - An element to search for. Refers to the tagName of DOM nodes.

JavaScript's `getElementsByTagName()` function is called to return the appropriate elements when this expression is used.

Example
Finds every DIV element.

```
$("div").css("border","9px solid red");
```

id
Selects a single element with the given id attribute.

Arguments
id - An ID to search for, specified via the id attribute of an element.

For id selectors, jQuery uses the JavaScript function `document.getElementById()`, which is extremely efficient. When another selector is attached to the id selector, such as `h2#pageTitle`, jQuery performs an additional check before identifying the element as a match.

As always, remember that as a developer, your time is typically the most valuable resource. Do not focus on optimization of selector speed unless it is clear that performance needs to be improved.

Each `id` value must be used only once within a document. If more than one element has been assigned the same ID, queries that use that ID will only select the first matched element in the DOM. This behavior should not be relied on, however; a document with more than one element using the same ID is invalid.

If the id contains characters like periods or colons you have to escape those characters with backslashes.

Example
Finds the element with the id "myDiv".

```
$("#myDiv").css("border","3px solid red");
```

Example
Finds the element with the id "myID.entry[1]". See how certain characters must be escaped with backslashes.

```
$("#myID\.entry\[1\]").css("border","3px solid red");
```

Basic Filter

focus
Selects element if it is currently focused.

As with other pseudo-class selectors (those that begin with a ":"), it is recommended to precede `:focus` with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':focus')` is equivalent to `$('*:focus')`. If you are looking for the currently focused element, `$( document.activeElement )` will retrieve it without having to search the whole DOM tree.

Example
Adds the focused class to whatever element has focus

```
$( "#content" ).delegate( "*", "focus blur", function( event ) {
  var elem = $( this );
  setTimeout(function() {
    elem.toggleClass( "focused", elem.is( ":focus" ) );
  }, 0);
});
```

animated
Select all elements that are in the progress of an animation at the time the selector is run.

Example
Change the color of any div that is animated.

```
$("#run").click(function(){
   $("div:animated").toggleClass("colored");
  });
  function animateIt() {
   $("#mover").slideToggle("slow", animateIt);
  }
  animateIt();
```

header
Selects all elements that are headers, like h1, h2, h3 and so on.

Example
Adds a background and text color to all the headers on the page.

```
$(":header").css({ background:'#CCC', color:'blue' });
```

lt
Select all elements at an index less than `index` within the matched set.

Arguments
index - Zero-based index.

index-related selectors

The index-related selectors (including this "less than" selector) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:lt(1)')` selects the first element in the document with the class `myclass`, rather than selecting no elements. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Example
Finds TDs less than the one with the 4th index (TD#4).

```
$("td:lt(4)").css("color", "red");
```

gt
Select all elements at an index greater than `index` within the matched set.

Arguments
index - Zero-based index.

index-related selectors

The index-related selector expressions (including this "greater than" selector) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:gt(1)')` selects elements after the second element in the document with the class `myclass`, rather than after the first. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Example
Finds TD #5 and higher. Reminder: the indexing starts at 0.

```
$("td:gt(4)").css("text-decoration", "line-through");
```

eq
Select the element at index n within the matched set.

Arguments
index - Zero-based index of the element to match.

The index-related selectors (`:eq()`, `:lt()`, `:gt()`, `:even`, `:odd`) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices `0` through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:eq(1)')` selects the second element in the document with the class myclass, rather than the first. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Unlike the .eq(index) method, the `:eq(index)` selector does *not* accept a negative value for `index`. For example, while `$('li').eq(-1)` selects the last `li` element, `$('li:eq(-1)')` selects nothing.

Example
Finds the third td.

```
$("td:eq(2)").css("color", "red");
```

Example
Apply three different styles to list items to demonstrate that `:eq()` is designed to select a single element while `:nth-child()` or `:eq()` within a looping construct such as `.each()` can select multiple elements.

```
// applies yellow background color to a single <li>
$("ul.nav li:eq(1)").css( "backgroundColor", "#ff0" );

// applies italics to text of the second <li> within each <ul class="nav">
$("ul.nav").each(function(index) {
 $(this).find("li:eq(1)").css( "fontStyle", "italic" );
});

// applies red text color to descendants of <ul class="nav">
// for each <li> that is the second child of its parent
$("ul.nav li:nth-child(2)").css( "color", "red" );
```

odd
Selects odd elements, zero-indexed. See also even.

In particular, note that the *0-based indexing* means that, counter-intuitively, `:odd` selects the second element, fourth element, and so on within the matched set.

Example
Finds odd table rows, matching the second, fourth and so on (index 1, 3, 5 etc.).

```
$("tr:odd").css("background-color", "#bbbbff");
```

even
Selects even elements, zero-indexed. See also odd.

In particular, note that the *0-based indexing* means that, counter-intuitively, `:even` selects the first element, third element, and so on within the matched set.

Example
Finds even table rows, matching the first, third and so on (index 0, 2, 4 etc.).

```
$("tr:even").css("background-color", "#bbbbff");
```

not
Selects all elements that do not match the given selector.

Arguments
selector - A selector with which to filter by.

All selectors are accepted inside `:not()`, for example: `:not(div a)` and `:not(div,a)`.

Additional Notes

The `.not()` method will end up providing you with more readable selections than pushing complex selectors or variables into a `:not()` selector filter. In most cases, it is a better choice.

Example
Finds all inputs that are not checked and highlights the next sibling span. Notice there is no change when clicking the checkboxes since no click events have been linked.

```
$("input:not(:checked) + span").css("background-color", "yellow");
 $("input").attr("disabled", "disabled");
```

last
Selects the last matched element.

Note that `:last` selects a single element by filtering the current jQuery collection and matching the last element within it.

Example
Finds the last table row.

```
$("tr:last").css({backgroundColor: 'yellow', fontWeight: 'bolder'});
```

first
Selects the first matched element.

The `:first` pseudo-class is equivalent to `:eq(0)`. It could also be written as `:lt(1)`. While this matches only a single element, :first-child can match more than one: One for each parent.

Example
Finds the first table row.

```
$("tr:first").css("font-style", "italic");
```

Child Filter

only-child
Selects all elements that are the only child of their parent.

If the parent has other child elements, nothing is matched.

Example
Change the text and add a border for each button that is the only child of its parent.

```
$("div button:only-child").text("Alone").css("border", "2px blue solid");
```

last-child
Selects all elements that are the last child of their parent.

While :last matches only a single element, `:last-child` can match more than one: one for each parent.

Example
Finds the last span in each matched div and adds some css plus a hover state.

```
$("div span:last-child")
    .css({color:"red", fontSize:"80%"})
    .hover(function () {
       $(this).addClass("solast");
      }, function () {
       $(this).removeClass("solast");
      });
```

first-child
Selects all elements that are the first child of their parent.

While :first matches only a single element, the `:first-child` selector can match more than one: one for each parent. This is equivalent to `:nth-child(1)`.

Example
Finds the first span in each matched div to underline and add a hover state.

```
$("div span:first-child")
    .css("text-decoration", "underline")
    .hover(function () {
       $(this).addClass("sogreen");
      }, function () {
       $(this).removeClass("sogreen");
      });
```

nth-child
Selects all elements that are the nth-child of their parent.

Arguments
index - The index of each child to match, starting with `1`, the string `even` or `odd`, or an equation (eg. `:nth-child(even)`, `:nth-child(4n)`)

Because jQuery's implementation of `:nth-child(n)` is strictly derived from the CSS specification, the value of n is "1-indexed", meaning that the counting starts at 1. For all other selector expressions, however, jQuery follows JavaScript's "0-indexed" counting. Therefore, given a single `<ul>` containing two `<li>`s, `$('li:nth-child(1)')` selects the first `<li>` while `$('li:eq(1)')` selects the second.

The `:nth-child(n)` pseudo-class is easily confused with `:eq(n)`, even though the two can result in dramatically different matched elements. With `:nth-child(n)`, all children are counted, regardless of what they are, and the specified element is selected only if it matches the selector attached to the pseudo-class. With `:eq(n)` only the selector attached to the pseudo-class is counted, not limited to children of any other element, and the (n+1)th one (n is 0-based) is selected.

Further discussion of this unusual usage can be found in the W3C CSS specification.

Example
Finds the second li in each matched ul and notes it.

```
$("ul li:nth-child(2)").append("<span> - 2nd!</span>");
```

Example
This is a playground to see how the selector works with different strings. Notice that this is different from the :even and :odd which have no regard for parent and just filter the list of elements to every other one. The :nth-child, however, counts the index of the child to its particular parent. In any case, it's easier to see than explain so...

```
$("button").click(function () {
   var str = $(this).text();
   $("tr").css("background", "white");
   $("tr" + str).css("background", "#ff0000");
   $("#inner").text(str);
  });
```

Content Filter

parent
Select all elements that are the parent of another element, including text nodes.

This is the inverse of `:empty`.

One important thing to note regarding the use of `:parent` (and `:empty`) is that child elements include text nodes.

The W3C recommends that the `<p>` element have at least one child node, even if that child is merely text (see http://www.w3.org/TR/html401/struct/text.html#edef-P). Some other elements, on the other hand, are empty (i.e. have no children) by definition: `<input>`, `<img>`, `<br>`, and `<hr>`, for example.

Example
Finds all tds with children, including text.

```
$("td:parent").fadeTo(1500, 0.3);
```

has
Selects elements which contain at least one element that matches the specified selector.

Arguments
selector - Any selector.

The expression `$('div:has(p)')` matches a `<div>` if a `<p>` exists anywhere among its descendants, not just as a direct child.

Example
Adds the class "test" to all divs that have a paragraph inside of them.

```
$("div:has(p)").addClass("test");
```

empty
Select all elements that have no children (including text nodes).

This is the inverse of `:parent`.

One important thing to note with :empty (and :parent) is that child elements include text nodes.

The W3C recommends that the <p> element have at least one child node, even if that child is merely text (see http://www.w3.org/TR/html401/struct/text.html#edef-P). Some other elements, on the other hand, are empty (i.e. have no children) by definition: <input>, <img>,
, and <hr>, for example.

Example
Finds all elements that are empty - they don't have child elements or text.

```
$("td:empty").text("Was empty!").css('background', 'rgb(255,220,200)');
```

contains
Select all elements that contain the specified text.

Arguments
text - A string of text to look for. It's case sensitive.

The matching text can appear directly within the selected element, in any of that element's descendants, or a combination thereof. As with attribute value selectors, text inside the parentheses of `:contains()` can be written as a bare word or surrounded by quotation marks. The text must have matching case to be selected.

Example
Finds all divs containing "John" and underlines them.

```
$("div:contains('John')").css("text-decoration", "underline");
```

Form

focus
Selects element if it is currently focused.

As with other pseudo-class selectors (those that begin with a ":"), it is recommended to precede `:focus` with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':focus')` is equivalent to `$('*:focus')`. If you are looking for the currently focused element, `$( document.activeElement )` will retrieve it without having to search the whole DOM tree.

Example
Adds the focused class to whatever element has focus

```
$( "#content" ).delegate( "*", "focus blur", function( event ) {
  var elem = $( this );
  setTimeout(function() {
    elem.toggleClass( "focused", elem.is( ":focus" ) );
  }, 0);
});
```

selected
Selects all elements that are selected.

The `:selected` selector works for `<option>` elements. It does not work for checkboxes or radio inputs; use `:checked` for them.

Example
Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.

```
$("select").change(function () {
     var str = "";
     $("select option:selected").each(function () {
        str += $(this).text() + " ";
       });
     $("div").text(str);
    })
    .trigger('change');
```

checked
Matches all elements that are checked.

The `:checked` selector works for checkboxes and radio buttons. For select elements, use the `:selected` selector.

Example
Finds all input elements that are checked.

```
function countChecked() {
 var n = $("input:checked").length;
 $("div").text(n + (n <= 1 ? " is" : " are") + " checked!");
}
countChecked();
$(":checkbox").click(countChecked);
```

Example

```
$("input").click(function() {
 $("#log").html( $(":checked").val() + " is checked!" );
});
```

disabled
Selects all elements that are disabled.

As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':disabled')` is equivalent to `$('*:disabled')`, so `$('input:disabled')` should be used instead.

Example
Finds all input elements that are disabled.

```
$("input:disabled").val("this is it");
```

enabled
Selects all elements that are enabled.

As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':enabled')` is equivalent to `$('*:enabled')`, so `$('input:enabled')` should be used instead.

Example

Finds all input elements that are enabled.

```
$("input:enabled").val("this is it");
```

file
Selects all elements of type file.

`:file` is equivalent to `[type="file"]`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':file')` is equivalent to `$('*:file')`, so `$('input:file')` should be used instead.

Example
Finds all file inputs.

```
var input = $("input:file").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

button
Selects all button elements and elements of type button.

An equivalent selector to `$(":button")` using valid CSS is `$("button, input[type='button']")`.

Example
Find all button inputs and mark them.

```
var input = $(":button").addClass("marked");
 $("div").text( "For this type jQuery found " + input.length + "." );
 $("form").submit(function () { return false; }); // so it won't submit
```

reset
Selects all elements of type reset.

`:reset` is equivalent to `[type="reset"]`

Example
Finds all reset inputs.

```
var input = $("input:reset").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

image
Selects all elements of type image.

`:image` is equivalent to `[type="image"]`

Example
Finds all image inputs.

```
var input = $("input:image").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
```

```
  $("form").submit(function () { return false; }); // so it won't submit
```

submit
Selects all elements of type submit.

The `:submit` selector typically applies to button or input elements. Note that some browsers treat `<button>` element as `type="default"` implicitly while others (such as Internet Explorer) do not.

Example
Finds all submit elements that are descendants of a td element.

```
var submitEl = $("td :submit")
   .parent('td')
   .css({background:"yellow", border:"3px red solid"})
  .end();

  $('#result').text('jQuery matched ' + submitEl.length + ' elements.');

  // so it won't submit
  $("form").submit(function () { return false; });

  // Extra JS to make the HTML easier to edit (None of this is relevant to the ':submit' selector
  $('#exampleTable').find('td').each(function(i, el) {
    var inputEl = $(el).children(),
      inputType = inputEl.attr('type') ? ' type="' + inputEl.attr('type') + '"' : '';
    $(el).before('<td>' + inputEl[0].nodeName + inputType + '</td>');
  })
```

checkbox
Selects all elements of type checkbox.

`$(':checkbox')` is equivalent to `$('[type=checkbox]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':checkbox')` is equivalent to `$('*:checkbox')`, so `$('input:checkbox')` should be used instead.

Example
Finds all checkbox inputs.

```
var input = $("form input:checkbox").wrap('<span></span>').parent().css({background:"yellow", border:"3px red
solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

radio
Selects all elements of type radio.

`$(':radio')` is equivalent to `$('[type=radio]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':radio')` is equivalent to `$('*:radio')`, so `$('input:radio')` should be used instead.

To select a set of associated radio buttons, you might use: `$('input[name=gender]:radio')`

Example
Finds all radio inputs.

```
var input = $("form input:radio").wrap('<span></span>').parent().css({background:"yellow", border:"3px red
```

```
solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

password

Selects all elements of type password.

`$(':password')` is equivalent to `$('[type=password]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':password')` is equivalent to `$('*:password')`, so `$('input:password')` should be used instead.

Example

Finds all password inputs.

```
var input = $("input:password").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

text

Selects all elements of type text.

`$(':text')` allows us to select all `<input type="text">` elements. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':text')` is equivalent to `$('*:text')`, so `$('input:text')` should be used instead.

Note: As of jQuery 1.5.2, `:text` selects `input` elements that have no specified `type` attribute (in which case `type="text"` is implied).

This difference in behavior between `$(':text')` and `$('[type=text]')`, can be seen below:

```
$('<input>').is('[type=text]'); // false
$('<input>').is(':text'); // true
```

Example

Finds all text inputs.

```
var input = $("form input:text").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

input

Selects all input, textarea, select and button elements.

The `:input` selector basically selects all form controls.

Example

Finds all input elements.

```
var allInputs = $(":input");
  var formChildren = $("form > *");
  $("#messages").text("Found " + allInputs.length + " inputs and the form has " +
              formChildren.length + " children.");

  // so it won't submit
```

```
$("form").submit(function () { return false; });
```

Hierarchy

next siblings

Selects all sibling elements that follow after the "prev" element, have the same parent, and match the filtering "siblings" selector.

Arguments

prev - Any valid selector.
siblings - A selector to filter elements that are the following siblings of the first selector.

The notable difference between (`prev + next`) and (`prev ~ siblings`) is their respective reach. While the former reaches only to the immediately following sibling element, the latter extends that reach to all following sibling elements.

Example

Finds all divs that are siblings after the element with #prev as its id. Notice the span isn't selected since it is not a div and the "niece" isn't selected since it is a child of a sibling, not an actual sibling.

```
$("#prev ~ div").css("border", "3px groove blue");
```

next adjacent

Selects all next elements matching "next" that are immediately preceded by a sibling "prev".

Arguments

prev - Any valid selector.
next - A selector to match the element that is next to the first selector.

One important point to consider with both the next adjacent sibling selector (`prev + next`) and the general sibling selector (`prev ~ siblings`) is that the elements on either side of the combinator must share the same parent.

Example

Finds all inputs that are next to a label.

```
$("label + input").css("color", "blue").val("Labeled!")
```

child

Selects all direct child elements specified by "child" of elements specified by "parent".

Arguments

parent - Any valid selector.
child - A selector to filter the child elements.

As a CSS selector, the child combinator is supported by all modern web browsers including Safari, Firefox, Opera, Chrome, and Internet Explorer 7 and above, but notably not by Internet Explorer versions 6 and below. However, in jQuery, this selector (along with all others) works across all supported browsers, including IE6.

The child combinator (E **>** F) can be thought of as a more specific form of the descendant combinator (E F) in that it selects only first-level descendants.

Note: The `$("> elem", context)` selector will be deprecated in a future release. Its usage is thus discouraged in lieu of using alternative selectors.

Example

Places a border around all list items that are children of <ul class="topnav"> .

```
$("ul.topnav > li").css("border", "3px double red");
```

descendant

Selects all elements that are descendants of a given ancestor.

Arguments
ancestor - Any valid selector.
descendant - A selector to filter the descendant elements.

A descendant of an element could be a child, grandchild, great-grandchild, and so on, of that element.

Example
Finds all input descendants of forms.

```
$("form input").css("border", "2px dotted blue");
```

jQuery Extensions

selected
Selects all elements that are selected.

The `:selected` selector works for `<option>` elements. It does not work for checkboxes or radio inputs; use `:checked` for them.

Example
Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.

```
$("select").change(function () {
     var str = "";
     $("select option:selected").each(function () {
        str += $(this).text() + " ";
       });
     $("div").text(str);
    })
    .trigger('change');
```

file
Selects all elements of type file.

`:file` is equivalent to `[type="file"]`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':file')` is equivalent to `$('*:file')`, so `$('input:file')` should be used instead.

Example
Finds all file inputs.

```
var input = $("input:file").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

button
Selects all button elements and elements of type button.

An equivalent selector to `$(":button")` using valid CSS is `$("button, input[type='button']")`.

Example
Find all button inputs and mark them.

```
var input = $(":button").addClass("marked");
 $("div").text( "For this type jQuery found " + input.length + "." );
 $("form").submit(function () { return false; }); // so it won't submit
```

reset
Selects all elements of type reset.

`:reset` is equivalent to `[type="reset"]`

Example
Finds all reset inputs.

```
var input = $("input:reset").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

image
Selects all elements of type image.

`:image` is equivalent to `[type="image"]`

Example
Finds all image inputs.

```
var input = $("input:image").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

submit
Selects all elements of type submit.

The `:submit` selector typically applies to button or input elements. Note that some browsers treat `<button>` element as `type="default"` implicitly while others (such as Internet Explorer) do not.

Example
Finds all submit elements that are descendants of a td element.

```
var submitEl = $("td :submit")
   .parent('td')
   .css({background:"yellow", border:"3px red solid"})
  .end();

  $('#result').text('jQuery matched ' + submitEl.length + ' elements.');

  // so it won't submit
  $("form").submit(function () { return false; });

  // Extra JS to make the HTML easier to edit (None of this is relevant to the ':submit' selector
  $('#exampleTable').find('td').each(function(i, el) {
    var inputEl = $(el).children(),
      inputType = inputEl.attr('type') ? ' type="' + inputEl.attr('type') + '"' : '';
    $(el).before('<td>' + inputEl[0].nodeName + inputType + '</td>');
  })
```

checkbox
Selects all elements of type checkbox.

`$(':checkbox')` is equivalent to `$('[type=checkbox]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':checkbox')` is equivalent to `$('*:checkbox')`, so `$('input:checkbox')` should be used instead.

Example
Finds all checkbox inputs.

```
var input = $("form input:checkbox").wrap('<span></span>').parent().css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

radio
Selects all elements of type radio.

`$(':radio')` is equivalent to `$('[type=radio]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':radio')` is equivalent to `$('*:radio')`, so `$('input:radio')` should be used instead.

To select a set of associated radio buttons, you might use: `$('input[name=gender]:radio')`

Example
Finds all radio inputs.

```
var input = $("form input:radio").wrap('<span></span>').parent().css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

password
Selects all elements of type password.

`$(':password')` is equivalent to `$('[type=password]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':password')` is equivalent to `$('*:password')`, so `$('input:password')` should be used instead.

Example
Finds all password inputs.

```
var input = $("input:password").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

text
Selects all elements of type text.

`$(':text')` allows us to select all `<input type="text">` elements. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':text')` is equivalent to `$('*:text')`, so `$('input:text')` should be used instead.

Note: As of jQuery 1.5.2, `:text` selects `input` elements that have no specified `type` attribute (in which case `type="text"` is implied).

This difference in behavior between `$(':text')` and `$('[type=text]')`, can be seen below:

```
$('<input>').is('[type=text]'); // false
$('<input>').is(':text'); // true
```

Example
Finds all text inputs.

```
var input = $("form input:text").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

input
Selects all input, textarea, select and button elements.

The `:input` selector basically selects all form controls.

Example
Finds all input elements.

```
var allInputs = $(":input");
  var formChildren = $("form > *");
  $("#messages").text("Found " + allInputs.length + " inputs and the form has " +
              formChildren.length + " children.");

  // so it won't submit
  $("form").submit(function () { return false; });
```

attributeNotEqual
Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector is equivalent to `:not([attr="value"])`.

Example
Finds all inputs that don't have the name 'newsletter' and appends text to the span next to it.

```
$('input[name!="newsletter"]').next().append('<b>; not newsletter</b>');
```

visible
Selects all elements that are visible.

Elements are considered visible if they consume space in the document. Visible elements have a width or height that is greater than zero.

Elements with `visibility: hidden` or `opacity: 0` are considered visible, since they still consume space in the layout. During animations that hide an element, the element is considered to be visible until the end of the animation. During animations to show an element, the element is considered to be visible at the start at the animation.

How `:visible` is calculated was changed in jQuery 1.3.2. The release notes outline the changes in more detail.

Example

Make all visible divs turn yellow on click.

```
$("div:visible").click(function () {
   $(this).css("background", "yellow");
  });
  $("button").click(function () {
   $("div:hidden").show("fast");
  });
```

hidden
Selects all elements that are hidden.

Elements can be considered hidden for several reasons:

- They have a CSS `display` value of `none`.
- They are form elements with `type="hidden"`.
- Their width and height are explicitly set to 0.
- An ancestor element is hidden, so the element is not shown on the page.

Elements with `visibility: hidden` or `opacity: 0` are considered to be visible, since they still consume space in the layout. During animations that hide an element, the element is considered to be visible until the end of the animation. During animations to show an element, the element is considered to be visible at the start of the animation.

How `:hidden` is determined was changed in jQuery 1.3.2. An element is assumed to be hidden if it or any of its parents consumes no space in the document. CSS visibility isn't taken into account (therefore `$(elem).css('visibility','hidden').is(':hidden') == false`). The release notes outline the changes in more detail.

Example
Shows all hidden divs and counts hidden inputs.

```
// in some browsers :hidden includes head, title, script, etc...
var hiddenEls = $("body").find(":hidden").not("script");

$("span:first").text("Found " + hiddenEls.length + " hidden elements total.");
$("div:hidden").show(3000);
$("span:last").text("Found " + $("input:hidden").length + " hidden inputs.");
```

parent
Select all elements that are the parent of another element, including text nodes.

This is the inverse of `:empty`.

One important thing to note regarding the use of `:parent` (and `:empty`) is that child elements include text nodes.

The W3C recommends that the <p> element have at least one child node, even if that child is merely text (see http://www.w3.org/TR/html401/struct/text.html#edef-P). Some other elements, on the other hand, are empty (i.e. have no children) by definition: `<input>`, `<img>`, `<br>`, and `<hr>`, for example.

Example
Finds all tds with children, including text.

```
$("td:parent").fadeTo(1500, 0.3);
```

has
Selects elements which contain at least one element that matches the specified selector.

Arguments
selector - Any selector.

The expression `$('div:has(p)')` matches a if a <p> exists anywhere among its descendants, not just as a direct child.

Example
Adds the class "test" to all divs that have a paragraph inside of them.

```
$("div:has(p)").addClass("test");
```

animated
Select all elements that are in the progress of an animation at the time the selector is run.

Example
Change the color of any div that is animated.

```
$("#run").click(function(){
   $("div:animated").toggleClass("colored");
  });
  function animateIt() {
   $("#mover").slideToggle("slow", animateIt);
  }
  animateIt();
```

header
Selects all elements that are headers, like h1, h2, h3 and so on.

Example
Adds a background and text color to all the headers on the page.

```
$(":header").css({ background:'#CCC', color:'blue' });
```

lt
Select all elements at an index less than `index` within the matched set.

Arguments
index - Zero-based index.

index-related selectors

The index-related selectors (including this "less than" selector) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:lt(1)')` selects the first element in the document with the class `myclass`, rather than selecting no elements. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Example
Finds TDs less than the one with the 4th index (TD#4).

```
$("td:lt(4)").css("color", "red");
```

gt
Select all elements at an index greater than `index` within the matched set.

Arguments
index - Zero-based index.

index-related selectors

The index-related selector expressions (including this "greater than" selector) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:gt(1)')` selects elements after the second element in the document with the class `myclass`, rather than after the first. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Example
Finds TD #5 and higher. Reminder: the indexing starts at 0.

```
$("td:gt(4)").css("text-decoration", "line-through");
```

eq
Select the element at index n within the matched set.

Arguments
index - Zero-based index of the element to match.

The index-related selectors (`:eq()`, `:lt()`, `:gt()`, `:even`, `:odd`) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices `0` through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:eq(1)')` selects the second element in the document with the class myclass, rather than the first. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Unlike the .eq(index) method, the `:eq(index)` selector does *not* accept a negative value for `index`. For example, while `$('li').eq(-1)` selects the last `li` element, `$('li:eq(-1)')` selects nothing.

Example
Finds the third td.

```
$("td:eq(2)").css("color", "red");
```

Example
Apply three different styles to list items to demonstrate that `:eq()` is designed to select a single element while `:nth-child()` or `:eq()` within a looping construct such as `.each()` can select multiple elements.

```
// applies yellow background color to a single <li>
$("ul.nav li:eq(1)").css( "backgroundColor", "#ff0" );

// applies italics to text of the second <li> within each <ul class="nav">
$("ul.nav").each(function(index) {
 $(this).find("li:eq(1)").css( "fontStyle", "italic" );
});

// applies red text color to descendants of <ul class="nav">
// for each <li> that is the second child of its parent
$("ul.nav li:nth-child(2)").css( "color", "red" );
```

odd
Selects odd elements, zero-indexed. See also even.

In particular, note that the *0-based indexing* means that, counter-intuitively, `:odd` selects the second element, fourth element, and so on within the matched set.

Example
Finds odd table rows, matching the second, fourth and so on (index 1, 3, 5 etc.).

```
$("tr:odd").css("background-color", "#bbbbff");
```

even
Selects even elements, zero-indexed. See also odd.

In particular, note that the *0-based indexing* means that, counter-intuitively, `:even` selects the first element, third element, and so on within the matched set.

Example
Finds even table rows, matching the first, third and so on (index 0, 2, 4 etc.).

```
$("tr:even").css("background-color", "#bbbbff");
```

last
Selects the last matched element.

Note that `:last` selects a single element by filtering the current jQuery collection and matching the last element within it.

Example
Finds the last table row.

```
$("tr:last").css({backgroundColor: 'yellow', fontWeight: 'bolder'});
```

first
Selects the first matched element.

The `:first` pseudo-class is equivalent to `:eq(0)`. It could also be written as `:lt(1)`. While this matches only a single element, :first-child can match more than one: One for each parent.

Example
Finds the first table row.

```
$("tr:first").css("font-style", "italic");
```

Visibility Filter

visible
Selects all elements that are visible.

Elements are considered visible if they consume space in the document. Visible elements have a width or height that is greater than zero.

Elements with `visibility: hidden` or `opacity: 0` are considered visible, since they still consume space in the layout. During animations that hide an element, the element is considered to be visible until the end of the animation. During animations to show an element, the element is considered to be visible at the start at the animation.

How `:visible` is calculated was changed in jQuery 1.3.2. The release notes outline the changes in more detail.

Example
Make all visible divs turn yellow on click.

```
$("div:visible").click(function () {
   $(this).css("background", "yellow");
 });
 $("button").click(function () {
```

```
  $("div:hidden").show("fast");
});
```

hidden

Selects all elements that are hidden.

Elements can be considered hidden for several reasons:

- They have a CSS `display` value of `none`.
- They are form elements with `type="hidden"`.
- Their width and height are explicitly set to 0.
- An ancestor element is hidden, so the element is not shown on the page.

Elements with `visibility: hidden` or `opacity: 0` are considered to be visible, since they still consume space in the layout. During animations that hide an element, the element is considered to be visible until the end of the animation. During animations to show an element, the element is considered to be visible at the start of the animation.

How `:hidden` is determined was changed in jQuery 1.3.2. An element is assumed to be hidden if it or any of its parents consumes no space in the document. CSS visibility isn't taken into account (therefore `$(elem).css('visibility','hidden').is(':hidden') == false`). The release notes outline the changes in more detail.

Example

Shows all hidden divs and counts hidden inputs.

```
// in some browsers :hidden includes head, title, script, etc...
var hiddenEls = $("body").find(":hidden").not("script");

$("span:first").text("Found " + hiddenEls.length + " hidden elements total.");
$("div:hidden").show(3000);
$("span:last").text("Found " + $("input:hidden").length + " hidden inputs.");
```

Traversing

each(function(index, Element))
Iterate over a jQuery object, executing a function for each matched element.

Arguments
function(index, Element) - A function to execute for each matched element.

The `.each()` method is designed to make DOM looping constructs concise and less error-prone. When called it iterates over the DOM elements that are part of the jQuery 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 keyword `this` refers to the element.

Suppose we had a simple unordered list on the page:

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

We can select the list items and iterate across them:

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

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

0: foo1: bar

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

Example
Iterates over three divs and sets their color property.

```
$(document.body).click(function () {
   $("div").each(function (i) {
    if (this.style.color != "blue") {
     this.style.color = "blue";
    } else {
     this.style.color = "";
    }
   });
  });
```

Example
If you want to have the jQuery object instead of the regular DOM element, use the $(this) function, for example:

```
$("span").click(function () {
   $("li").each(function(){
    $(this).toggleClass("example");
   });
  });
```

Example
You can use 'return' to break out of each() loops early.

```
$("button").click(function () {
   $("div").each(function (index, domEle) {
    // domEle == this
```

```
    $(domEle).css("backgroundColor", "yellow");
    if ($(this).is("#stop")) {
     $("span").text("Stopped at div index #" + index);
     return false;
    }
   });
  });
```

Filtering

has(selector)

Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

Arguments

selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.has()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against the descendants of the matching elements; the element will be included in the result if any of its descendant elements matches the selector.

Consider a page with a nested list as follows:

```
 <ul>
 <li>list item 1</li>
 <li>list item 2
  <ul>
   <li>list item 2-a</li>
   <li>list item 2-b</li>
  </ul>
 </li>
 <li>list item 3</li>
 <li>list item 4</li>
</ul>
```

We can apply this method to the set of list items as follows:

```
$('li').has('ul').css('background-color', 'red');
```

The result of this call is a red background for item 2, as it is the only `<li>` that has a `<ul>` among its descendants.

Example

Check if an element is inside another.

```
$("ul").append("<li>" + ($("ul").has("li").length ? "Yes" : "No") + "</li>");
 $("ul").has("li").addClass("full");
```

first()

Reduce the set of matched elements to the first in the set.

[

Given a jQuery object that represents a set of DOM elements, the `.first()` method constructs a new jQuery object from the first matching element.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
```

```
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').first().css('background-color', 'red');
```

The result of this call is a red background for the first item.

Example
Highlight the first span in a paragraph.

```
$("p span").first().addClass('highlight');
```

last()
Reduce the set of matched elements to the final one in the set.

[

Given a jQuery object that represents a set of DOM elements, the `.last()` method constructs a new jQuery object from the last matching element.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').last().css('background-color', 'red');
```

The result of this call is a red background for the final item.

Example
Highlight the last span in a paragraph.

```
$("p span").last().addClass('highlight');
```

slice(start, [end])
Reduce the set of matched elements to a subset specified by a range of indices.

Arguments
start - An integer indicating the 0-based position at which the elements begin to be selected. If negative, it indicates an offset from the end of the set.
end - An integer indicating the 0-based position at which the elements stop being selected. If negative, it indicates an offset from the end of the set. If omitted, the range continues until the end of the set.

Given a jQuery object that represents a set of DOM elements, the `.slice()` method constructs a new jQuery object from a subset of the matching elements. The supplied `start` index identifies the position of one of the elements in the set; if `end` is omitted, all elements after this one will be included in the result.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').slice(2).css('background-color', 'red');
```

The result of this call is a red background for items 3, 4, and 5. Note that the supplied index is zero-based, and refers to the position of elements within the jQuery object, not within the DOM tree.

The end parameter allows us to limit the selected range even further. For example:

```
$('li').slice(2, 4).css('background-color', 'red');
```

Now only items 3 and 4 are selected. The index is once again zero-based; the range extends up to but not including the specified index.

Negative Indices

The jQuery `.slice()` method is patterned after the JavaScript .slice() method for arrays. One of the features that it mimics is the ability for negative numbers to be passed as either the `start` or `end` parameter. If a negative number is provided, this indicates a position starting from the end of the set, rather than the beginning. For example:

```
$('li').slice(-2, -1).css('background-color', 'red');
```

This time only list item 4 is turned red, since it is the only item in the range between two from the end (`-2`) and one from the end (`-1`).

Example

Turns divs yellow based on a random slice.

```
function colorEm() {
   var $div = $("div");
   var start = Math.floor(Math.random() *
               $div.length);
   var end = Math.floor(Math.random() *
             ($div.length - start)) +
             start + 1;
   if (end == $div.length) end = undefined;
   $div.css("background", "");
   if (end)
    $div.slice(start, end).css("background", "yellow");
    else
    $div.slice(start).css("background", "yellow");

   $("span").text('$("div").slice(' + start +
           (end ? ', ' + end : '') +
           ').css("background", "yellow");');
  }

  $("button").click(colorEm);
```

Example

Selects all paragraphs, then slices the selection to include only the first element.

```
$("p").slice(0, 1).wrapInner("<b></b>");
```

Example

Selects all paragraphs, then slices the selection to include only the first and second element.

```
$("p").slice(0, 2).wrapInner("<b></b>");
```

Example
Selects all paragraphs, then slices the selection to include only the second element.

```
$("p").slice(1, 2).wrapInner("<b></b>");
```

Example
Selects all paragraphs, then slices the selection to include only the second and third element.

```
$("p").slice(1).wrapInner("<b></b>");
```

Example
Selects all paragraphs, then slices the selection to include only the third element.

```
$("p").slice(-1).wrapInner("<b></b>");
```

not(selector)
Remove elements from the set of matched elements.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.not()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; the elements that don't match the selector will be included in the result.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').not(':even').css('background-color', 'red');
```

The result of this call is a red background for items 2 and 4, as they do not match the selector (recall that :even and :odd use 0-based indexing).

Removing Specific Elements

The second version of the `.not()` method allows us to remove elements from the matched set, assuming we have found those elements previously by some other means. For example, suppose our list had an id applied to one of its items:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li id="notli">list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can fetch the third list item using the native JavaScript `getElementById()` function, then remove it from a jQuery object:

```
$('li').not(document.getElementById('notli'))
 .css('background-color', 'red');
```

This statement changes the color of items 1, 2, 4, and 5. We could have accomplished the same thing with a simpler jQuery expression, but this technique can be useful when, for example, other libraries provide references to plain DOM nodes.

As of jQuery 1.4, the `.not()` method can take a function as its argument in the same way that `.filter()` does. Elements for which the function returns `true` are excluded from the filtered set; all other elements are included.

Example
Adds a border to divs that are not green or blue.

```
$("div").not(".green, #blueone")
      .css("border-color", "red");
```

Example
Removes the element with the ID "selected" from the set of all paragraphs.

```
$("p").not( $("#selected")[0] )
```

Example
Removes the element with the ID "selected" from the set of all paragraphs.

```
$("p").not("#selected")
```

Example
Removes all elements that match "div p.selected" from the total set of all paragraphs.

```
$("p").not($("div p.selected"))
```

map(callback(index, domElement))
Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.

Arguments
callback(index, domElement) - A function object that will be invoked for each element in the current set.

As the return value is a jQuery-wrapped array, it's very common to `get()` the returned object to work with a basic array.

The `.map()` method is particularly useful for getting or setting the value of a collection of elements. Consider a form with a set of checkboxes in it:

```
<form method="post" action="">
 <fieldset>
  <div>
   <label for="two">2</label>
   <input type="checkbox" value="2" id="two" name="number[]">
  </div>
  <div>
   <label for="four">4</label>
   <input type="checkbox" value="4" id="four" name="number[]">
  </div>
  <div>
   <label for="six">6</label>
   <input type="checkbox" value="6" id="six" name="number[]">
  </div>
  <div>
   <label for="eight">8</label>
   <input type="checkbox" value="8" id="eight" name="number[]">
```

```
  </div>
 </fieldset>
</form>
```

We can get a comma-separated list of checkbox IDs:

```
$(':checkbox').map(function() {
 return this.id;
}).get().join(',');
```

The result of this call is the string, "two,four,six,eight".

Within the callback function, this refers to the current DOM element for each iteration. The function can return an individual data item or an array of data items to be inserted into the resulting set. If an array is returned, the elements inside the array are inserted into the set. If the function returns null or undefined, no element will be inserted.

Example

Build a list of all the values within a form.

```
$("p").append( $("input").map(function(){
   return $(this).val();
  }).get().join(", ") );
```

Example

A contrived example to show some functionality.

```
var mappedItems = $("li").map(function (index) {
 var replacement = $("<li>").text($(this).text()).get(0);
 if (index == 0) {
  /* make the first item all caps */
  $(replacement).text($(replacement).text().toUpperCase());
 } else if (index == 1 || index == 3) {
  /* delete the second and fourth items */
  replacement = null;
 } else if (index == 2) {
  /* make two of the third item and add some text */
  replacement = [replacement,$("<li>").get(0)];
  $(replacement[0]).append("<b> - A</b>");
  $(replacement[1]).append("Extra <b> - B</b>");
 }

 /* replacement will be a dom element, null,
   or an array of dom elements */
 return replacement;
});
$("#results").append(mappedItems);
```

Example

Equalize the heights of the divs.

```
$.fn.equalizeHeights = function() {
 var maxHeight = this.map(function(i,e) {
  return $(e).height();
 }).get();

 return this.height( Math.max.apply(this, maxHeight) );
};

$('input').click(function(){
 $('div').equalizeHeights();
});
```

is(selector)

Check the current matched set of elements against a selector, element, or jQuery object and return `true` if at least one of these elements matches the given arguments.

Arguments

selector - A string containing a selector expression to match elements against.

Unlike other filtering methods, `.is()` does not create a new jQuery object. Instead, it allows you to test the contents of a jQuery object without modification. This is often useful inside callbacks, such as event handlers.

Suppose you have a list, with two of its items containing a child element:

```
<ul>
 <li>list <strong>item 1</strong></li>
 <li><span>list item 2</span></li>
 <li>list item 3</li>
</ul>
```

You can attach a click handler to the <ul> element, and then limit the code to be triggered only when a list item itself, not one of its children, is clicked:

```
$("ul").click(function(event) {
 var $target = $(event.target);
 if ( $target.is("li") ) {
  $target.css("background-color", "red");
 }
});
```

Now, when the user clicks on the word "list" in the first item or anywhere in the third item, the clicked list item will be given a red background. However, when the user clicks on item 1 in the first item or anywhere in the second item, nothing will occur, because in those cases the target of the event would be `<strong>` or `<span>`, respectively.

Prior to jQuery 1.7, in selector strings with positional selectors such as `:first`, `:gt()`, or `:even`, the positional filtering is done against the jQuery object passed to `.is()`, *not* against the containing document. So for the HTML shown above, an expression such as `$("li:first").is("li:last")` returns `true`, but `$("li:first-child").is("li:last-child")` returns `false`. In addition, a bug in Sizzle prevented many positional selectors from working properly. These two factors made positional selectors almost unusable in filters.

Starting with jQuery 1.7, selector strings with positional selectors apply the selector against the document, and then determine whether the first element of the current jQuery set matches any of the resulting elements. So for the HTML shown above, an expression such as `$("li:first").is("li:last")` returns `false`. Note that since positional selectors are jQuery additions and not W3C standard, we recommend using the W3C selectors whenever feasible.

Using a Function

The second form of this method evaluates expressions related to elements based on a function rather than a selector. For each element, if the function returns `true`, `.is()` returns `true` as well. For example, given a somewhat more involved HTML snippet:

```
<ul>
 <li><strong>list</strong> item 1 - one strong tag</li>
 <li><strong>list</strong> item <strong>2</strong> -
  two <span>strong tags</span></li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

You can attach a click handler to every `<li>` that evaluates the number of `<strong>` elements within the clicked `<li>` at that time like so:

```
$("li").click(function() {
 var $li = $(this),
  isWithTwo = $li.is(function() {
   return $('strong', this).length === 2;
  });
 if ( isWithTwo ) {
  $li.css("background-color", "green");
 } else {
  $li.css("background-color", "red");
 }
});
```

Example
Shows a few ways is() can be used inside an event handler.

```
$("div").one('click', function () {
  if ($(this).is(":first-child")) {
   $("p").text("It's the first div.");
  } else if ($(this).is(".blue,.red")) {
   $("p").text("It's a blue or red div.");
  } else if ($(this).is(":contains('Peter')")) {
   $("p").text("It's Peter!");
  } else {
   $("p").html("It's nothing <em>special</em>.");
  }
  $("p").hide().slideDown("slow");
  $(this).css({"border-style": "inset", cursor:"default"});
 });
```

Example
Returns true, because the parent of the input is a form element.

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
 $("div").text("isFormParent = " + isFormParent);
```

Example
Returns false, because the parent of the input is a p element.

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
 $("div").text("isFormParent = " + isFormParent);
```

Example
Checks against an existing collection of alternating list elements. Blue, alternating list elements slide up while others turn red.

```
var $alt = $("#browsers li:nth-child(2n)").css("background", "#00FFFF");
 $('li').click(function() {
  var $li = $(this);
  if ( $li.is( $alt ) ) {
   $li.slideUp();
  } else {
   $li.css("background", "red");
  }
 });
```

Example
An alternate way to achieve the above example using an element rather than a jQuery object. Checks against an existing collection of alternating list elements. Blue, alternating list elements slide up while others turn red.

```
var $alt = $("#browsers li:nth-child(2n)").css("background", "#00FFFF");
 $('li').click(function() {
  if ( $alt.is( this ) ) {
   $(this).slideUp();
```

```
  } else {
   $(this).css("background", "red");
  }
 });
```

eq(index)

Reduce the set of matched elements to the one at the specified index.

Arguments

index - An integer indicating the 0-based position of the element.

Given a jQuery object that represents a set of DOM elements, the `.eq()` method constructs a new jQuery object from one element within that set. The supplied index identifies the position of this element in the set.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').eq(2).css('background-color', 'red');
```

The result of this call is a red background for item 3. Note that the supplied index is zero-based, and refers to the position of the element within the jQuery object, not within the DOM tree.

Providing a negative number indicates a position starting from the end of the set, rather than the beginning. For example:

```
$('li').eq(-2).css('background-color', 'red');
```

This time list item 4 is turned red, since it is two from the end of the set.

If an element cannot be found at the specified zero-based index, the method constructs a new jQuery object with an empty set and a `length` property of 0.

```
$('li').eq(5).css('background-color', 'red');
```

Here, none of the list items is turned red, since `.eq(5)` indicates the sixth of five list items.

Example

Turn the div with index 2 blue by adding an appropriate class.

```
$("body").find("div").eq(2).addClass("blue");
```

filter(selector)

Reduce the set of matched elements to those that match the selector or pass the function's test.

Arguments

selector - A string containing a selector expression to match the current set of elements against.

Given a jQuery object that represents a set of DOM elements, the `.filter()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; all elements matching the selector will be included in the result.

Consider a page with a simple list on it: list item 1 list item 2 list item 3 list item 4 list item 5 list item 6

We can apply this method to the set of list items:

```
 $('li').filter(':even').css('background-color', 'red');
```

The result of this call is a red background for items 1, 3, and 5, as they match the selector (recall that `:even` and `:odd` use 0-based indexing).

Using a Filter Function

The second form of this method allows us to filter elements against a function rather than a selector. For each element, if the function returns `true` (or a "truthy" value), the element will be included in the filtered set; otherwise, it will be excluded. Suppose we have a somewhat more involved HTML snippet:

```
<ul>
 <li><strong>list</strong> item 1 -
  one strong tag</li>
 <li><strong>list</strong> item <strong>2</strong> -
  two <span>strong tags</span></li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
 <li>list item 6</li>
</ul>
```

We can select the list items, then filter them based on their contents:

```
$('li').filter(function(index) {
 return $('strong', this).length == 1;
}).css('background-color', 'red');
```

This code will alter the first list item only, as it contains exactly one `<strong>` tag. Within the filter function, `this` refers to each DOM element in turn. The parameter passed to the function tells us the index of that DOM element within the set matched by the jQuery object.

We can also take advantage of the `index` passed through the function, which indicates the 0-based position of the element within the unfiltered set of matched elements:

```
$('li').filter(function(index) {
 return index % 3 == 2;
}).css('background-color', 'red');
```

This alteration to the code will cause the third and sixth list items to be highlighted, as it uses the modulus operator (%) to select every item with an `index` value that, when divided by 3, has a remainder of 2.

Example

Change the color of all divs; then add a border to those with a "middle" class.

```
$("div").css("background", "#c8ebcc")
```

```
      .filter(".middle")
      .css("border-color", "red");
```

Example
Change the color of all divs; then add a border to the second one (index == 1) and the div with an id of "fourth."

```
$("div").css("background", "#b4b0da")
      .filter(function (index) {
         return index == 1 || $(this).attr("id") == "fourth";
        })
      .css("border", "3px double red");
```

Example
Select all divs and filter the selection with a DOM element, keeping only the one with an id of "unique".

```
$("div").filter( document.getElementById("unique") )
```

Example
Select all divs and filter the selection with a jQuery object, keeping only the one with an id of "unique".

```
$("div").filter( $("#unique") )
```

Miscellaneous Traversing

end()
End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

Most of jQuery's DOM traversal methods operate on a jQuery object instance and produce a new one, matching a different set of DOM elements. When this happens, it is as if the new set of elements is pushed onto a stack that is maintained inside the object. Each successive filtering method pushes a new element set onto the stack. If we need an older element set, we can use `end()` to pop the sets back off of the stack.

Suppose we have a couple short lists on a page:

```
<ul class="first">
  <li class="foo">list item 1</li>
  <li>list item 2</li>
  <li class="bar">list item 3</li>
</ul>
<ul class="second">
  <li class="foo">list item 1</li>
  <li>list item 2</li>
  <li class="bar">list item 3</li>
</ul>
```

The `end()` method is useful primarily when exploiting jQuery's chaining properties. When not using chaining, we can usually just call up a previous object by variable name, so we don't need to manipulate the stack. With `end()`, though, we can string all the method calls together:

```
$('ul.first').find('.foo').css('background-color', 'red')
  .end().find('.bar').css('background-color', 'green');
```

This chain searches for items with the class `foo` within the first list only and turns their backgrounds red. Then `end()` returns the object to its state before the call to `find()`, so the second `find()` looks for '.bar' inside `<ul class="first">`, not just inside that list's `<li class="foo">`, and turns the matching elements' backgrounds green. The net result is that items 1 and 3 of the first list have a colored background, and none of the items from the second list do.

A long jQuery chain can be visualized as a structured code block, with filtering methods providing the openings of nested blocks and `end()` methods

closing them:

```
$('ul.first').find('.foo')
 .css('background-color', 'red')
.end().find('.bar')
 .css('background-color', 'green')
.end();
```

The last `end()` is unnecessary, as we are discarding the jQuery object immediately thereafter. However, when the code is written in this form, the `end()` provides visual symmetry and a sense of completion -making the program, at least to the eyes of some developers, more readable, at the cost of a slight hit to performance as it is an additional function call.

Example
Selects all paragraphs, finds span elements inside these, and reverts the selection back to the paragraphs.

```
jQuery.fn.showTags = function (n) {
   var tags = this.map(function () {
               return this.tagName;
              })
            .get().join(", ");
   $("b:eq(" + n + ")").text(tags);
   return this;
  };

  $("p").showTags(0)
     .find("span")
     .showTags(1)
     .css("background", "yellow")
     .end()
     .showTags(2)
     .css("font-style", "italic");
```

Example
Selects all paragraphs, finds span elements inside these, and reverts the selection back to the paragraphs.

```
$("p").find("span").end().css("border", "2px red solid");
```

andSelf()
Add the previous set of elements on the stack to the current set.

As described in the discussion for `.end()`, jQuery objects maintain an internal stack that keeps track of changes to the matched set of elements. When one of the DOM traversal methods is called, the new set of elements is pushed onto the stack. If the previous set of elements is desired as well, `.andSelf()` can help.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

The result of the following code is a red background behind items 3, 4 and 5:

```
$('li.third-item').nextAll().andSelf()
```

```
 .css('background-color', 'red');
```

First, the initial selector locates item 3, initializing the stack with the set containing just this item. The call to `.nextAll()` then pushes the set of items 4 and 5 onto the stack. Finally, the `.andSelf()` invocation merges these two sets together, creating a jQuery object that points to all three items in document order: `{[<li.third-item>,<li>,<li> ]}`.

Example
Find all `div`s, and all the paragraphs inside of them, and give them both class names. Notice the `div` doesn't have the yellow background color since it didn't use `.andSelf()`.

```
$("div").find("p").andSelf().addClass("border");
  $("div").find("p").addClass("background");
```

contents()
Get the children of each element in the set of matched elements, including text and comment nodes.

Given a jQuery object that represents a set of DOM elements, the `.contents()` method allows us to search through the immediate children of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.contents()` and `.children()` methods are similar, except that the former includes text nodes as well as HTML elements in the resulting jQuery object.

The `.contents()` method can also be used to get the content document of an iframe, if the iframe is on the same domain as the main page.

Consider a simple `<div>` with a number of text nodes, each of which is separated by two line break elements (`<br />`):

```
<div class="container">
 Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
 do eiusmod tempor incididunt ut labore et dolore magna aliqua.
 <br /><br />
 Ut enim ad minim veniam, quis nostrud exercitation ullamco
 laboris nisi ut aliquip ex ea commodo consequat.
 <br /> <br />
 Duis aute irure dolor in reprehenderit in voluptate velit
 esse cillum dolore eu fugiat nulla pariatur.
</div>
```

We can employ the `.contents()` method to help convert this blob of text into three well-formed paragraphs:

```
$('.container').contents().filter(function() {
 return this.nodeType == 3;
})
 .wrap('<p></p>')
.end()
.filter('br')
 .remove();
```

This code first retrieves the contents of `<div class="container">` and then filters it for text nodes, which are wrapped in paragraph tags. This is accomplished by testing the .nodeType property of the element. This DOM property holds a numeric code indicating the node's type; text nodes use the code 3. The contents are again filtered, this time for `<br />` elements, and these elements are removed.

Example
Find all the text nodes inside a paragraph and wrap them with a bold tag.

```
$("p").contents().filter(function(){ return this.nodeType != 1; }).wrap("<b/>");
```

Example
Change the background colour of links inside of an iframe.

```
$("#frameDemo").contents().find("a").css("background-color","#BADA55");
```

add(selector)

Add elements to the set of matched elements.

Arguments

selector - A string representing a selector expression to find additional elements to add to the set of matched elements.

Given a jQuery object that represents a set of DOM elements, the `.add()` method constructs a new jQuery object from the union of those elements and the ones passed into the method. The argument to `.add()` can be pretty much anything that `$()` accepts, including a jQuery selector expression, references to DOM elements, or an HTML snippet.

The updated set of elements can be used in a following (chained) method, or assigned to a variable for later use. For example:

```
$("p").add("div").addClass("widget");
var pdiv = $("p").add("div");
```

The following will *not* save the added elements, because the `.add()` method creates a new set and leaves the original set in pdiv unchanged:

```
var pdiv = $("p");
pdiv.add("div"); // WRONG, pdiv will not change
```

Consider a page with a simple list and a paragraph following it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
</ul>
<p>a paragraph</p>
```

We can select the list items and then the paragraph by using either a selector or a reference to the DOM element itself as the `.add()` method's argument:

```
$('li').add('p').css('background-color', 'red');
```

Or:

```
$('li').add(document.getElementsByTagName('p')[0])
 .css('background-color', 'red');
```

The result of this call is a red background behind all four elements.Using an HTML snippet as the `.add()` method's argument (as in the third version), we can create additional elements on the fly and add those elements to the matched set of elements. Let's say, for example, that we want to alter the background of the list items along with a newly created paragraph:

```
$('li').add('<p id="new">new paragraph</p>')
 .css('background-color', 'red');
```

Although the new paragraph has been created and its background color changed, it still does not appear on the page. To place it on the page, we could add one of the insertion methods to the chain.

As of jQuery 1.4 the results from .add() will always be returned in document order (rather than a simple concatenation).

Note: To reverse the `.add()` you can use .not(elements | selector) to remove elements from the jQuery results, or .end() to return to the selection before you added.

Example

Finds all divs and makes a border. Then adds all paragraphs to the jQuery object to set their backgrounds yellow.

```
$("div").css("border", "2px solid red")
    .add("p")
    .css("background", "yellow");
```

Example
Adds more elements, matched by the given expression, to the set of matched elements.

```
$("p").add("span").css("background", "yellow");
```

Example
Adds more elements, created on the fly, to the set of matched elements.

```
$("p").clone().add("<span>Again</span>").appendTo(document.body);
```

Example
Adds one or more Elements to the set of matched elements.

```
$("p").add(document.getElementById("a")).css("background", "yellow");
```

Example
Demonstrates how to add (or push) elements to an existing collection

```
var collection = $("p");
// capture the new collection
collection = collection.add(document.getElementById("a"));
collection.css("background", "yellow");
```

not(selector)
Remove elements from the set of matched elements.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.not()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; the elements that don't match the selector will be included in the result.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').not(':even').css('background-color', 'red');
```

The result of this call is a red background for items 2 and 4, as they do not match the selector (recall that :even and :odd use 0-based indexing).

Removing Specific Elements

The second version of the `.not()` method allows us to remove elements from the matched set, assuming we have found those elements previously by some other means. For example, suppose our list had an id applied to one of its items:

```
<ul>
```

```
 <li>list item 1</li>
 <li>list item 2</li>
 <li id="notli">list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can fetch the third list item using the native JavaScript `getElementById()` function, then remove it from a jQuery object:

```
$('li').not(document.getElementById('notli'))
 .css('background-color', 'red');
```

This statement changes the color of items 1, 2, 4, and 5. We could have accomplished the same thing with a simpler jQuery expression, but this technique can be useful when, for example, other libraries provide references to plain DOM nodes.

As of jQuery 1.4, the `.not()` method can take a function as its argument in the same way that `.filter()` does. Elements for which the function returns `true` are excluded from the filtered set; all other elements are included.

Example
Adds a border to divs that are not green or blue.

```
$("div").not(".green, #blueone")
      .css("border-color", "red");
```

Example
Removes the element with the ID "selected" from the set of all paragraphs.

```
$("p").not( $("#selected")[0] )
```

Example
Removes the element with the ID "selected" from the set of all paragraphs.

```
$("p").not("#selected")
```

Example
Removes all elements that match "div p.selected" from the total set of all paragraphs.

```
$("p").not($("div p.selected"))
```

Tree Traversal

parentsUntil([selector], [filter])
Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

Arguments
selector - A string containing a selector expression to indicate where to stop matching ancestor elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.parentsUntil()` method traverses through the ancestors of these elements until it reaches an element matched by the selector passed in the method's argument. The resulting jQuery object contains all of the ancestors up to but not including the one matched by the `.parentsUntil()` selector.

If the selector is not matched or is not supplied, all ancestors will be selected; in these cases it selects the same elements as the `.parents()` method does when no selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be used for the first **.parentsUntil()** argument.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

Example
Find the ancestors of <li class="item-a"> up to <ul class="level-1"> and give them a red background color. Also, find ancestors of <li class="item-2"> that have a class of "yes" up to <ul class="level-1"> and give them a green border.

```
$("li.item-a").parentsUntil(".level-1")
 .css("background-color", "red");

$("li.item-2").parentsUntil( $("ul.level-1"), ".yes" )
 .css("border", "3px solid green");
```

prevUntil([selector], [filter])
Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

Arguments
selector - A string containing a selector expression to indicate where to stop matching preceding sibling elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.prevUntil()` method searches through the predecessors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all previous siblings up to but not including the one matched by the `.prevUntil()` selector; the elements are returned in order from the closest sibling to the farthest.

If the selector is not matched or is not supplied, all previous siblings will be selected; in these cases it selects the same elements as the `.prevAll()` method does when no filter selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be used for the first **.prevUntil()** argument.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

Example
Find the siblings that precede <dt id="term-2"> up to the preceding <dt> and give them a red background color. Also, find previous <dd> siblings of <dt id="term-3"> up to <dt id="term-1"> and give them a green text color.

```
$("#term-2").prevUntil("dt")
 .css("background-color", "red");

var term1 = document.getElementById('term-1');
$("#term-3").prevUntil(term1, "dd")
 .css("color", "green");
```

nextUntil([selector], [filter])
Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

Arguments
selector - A string containing a selector expression to indicate where to stop matching following sibling elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.nextUntil()` method searches through the successors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all following siblings up to but not including the one matched by the `.nextUntil()` argument.

If the selector is not matched or is not supplied, all following siblings will be selected; in these cases it selects the same elements as the `.nextAll()` method does when no filter selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be passed to the `.nextUntil()` method.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing

whether they match it.

Example
Find the siblings that follow <dt id="term-2"> up to the next <dt> and give them a red background color. Also, find <dd> siblings that follow <dt id="term-1"> up to <dt id="term-3"> and give them a green text color.

```
$("#term-2").nextUntil("dt")
 .css("background-color", "red");

var term3 = document.getElementById("term-3");
$("#term-1").nextUntil(term3, "dd")
 .css("color", "green");
```

siblings([selector])
Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.siblings()` method allows us to search through the siblings of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find its siblings:

```
$('li.third-item').siblings().css('background-color', 'red');
```

The result of this call is a red background behind items 1, 2, 4, and 5. Since we do not supply a selector expression, all of the siblings are part of the object. If we had supplied one, only the matching items among these four would be included.

The original element is not included among the siblings, which is important to remember when we wish to find all elements at a particular level of the DOM tree.

Example
Find the unique siblings of all yellow li elements in the 3 lists (including other yellow li elements if appropriate).

```
var len = $(".hilite").siblings()
            .css("color", "red")
            .length;
  $("b").text(len);
```

Example
Find all siblings with a class "selected" of each div.

```
$("p").siblings(".selected").css("background", "yellow");
```

prevAll([selector])

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.prevAll()` method searches through the predecessors of these elements in the DOM tree and construct a new jQuery object from the matching elements; the elements are returned in order beginning with the closest sibling.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the elements which come before it:

```
$('li.third-item').prevAll().css('background-color', 'red');
```

The result of this call is a red background behind items 1 and 2. Since we do not supply a selector expression, these preceding elements are unequivocally included as part of the object. If we had supplied one, the elements would be tested for a match before they were included.

Example
Locate all the divs preceding the last div and give them a class.

```
$("div:last").prevAll().addClass("before");
```

prev([selector])
Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.prev()` method searches for the predecessor of each of these elements in the DOM tree and constructs a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that can be passed to the `$()` function. If the selector is supplied, the preceding element will be filtered by testing whether it match the selector.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

To select the element that comes immediately before item three:

```
$('li.third-item').prev().css('background-color', 'red');
```

The result of this call is a red background behind item 2. Since no selector expression is supplied, this preceding element is unequivocally included as part of the object. If one had been supplied, the element would be tested for a match before it was included.

If no previous sibling exists, or if the previous sibling element does not match a supplied selector, an empty jQuery object is returned.

To select *all* preceding sibling elements, rather than just the preceding *adjacent* sibling, use the .prevAll() method.

Example
Find the very previous sibling of each div.

```
var $curr = $("#start");
  $curr.css("background", "#f99");
  $("button").click(function () {
   $curr = $curr.prev();
   $("div").css("background", "");
   $curr.css("background", "#f99");
  });
```

Example
For each paragraph, find the very previous sibling that has a class "selected".

```
$("p").prev(".selected").css("background", "yellow");
```

parents([selector])
Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.parents()` method allows us to search through the ancestors of these elements in the DOM tree and construct a new jQuery object from the matching elements ordered from immediate parent on up; the elements are returned in order from the closest parent to the outer ones. The `.parents()` and `.parent()` methods are similar, except that the latter only travels a single level up the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at item A, we can find its ancestors:

```
$('li.item-a').parents().css('background-color', 'red');
```

The result of this call is a red background for the level-2 list, item II, and the level-1 list (and on up the DOM tree all the way to the <html> element). Since we do not supply a selector expression, all of the ancestors are part of the returned jQuery object. If we had supplied one, only the matching items among these would be included.

Example
Find all parent elements of each b.

```
var parentEls = $("b").parents()
      .map(function () {
         return this.tagName;
        })
      .get().join(", ");
$("b").append("<strong>" + parentEls + "</strong>");
```

Example
Click to find all unique div parent elements of each span.

```
function showParents() {
 $("div").css("border-color", "white");
 var len = $("span.selected")
          .parents("div")
          .css("border", "2px red solid")
          .length;
 $("b").text("Unique div parents: " + len);
}
$("span").click(function () {
 $(this).toggleClass("selected");
 showParents();
});
```

parent([selector])
Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the .parent() method allows us to search through the parents of these elements in the DOM tree and construct a new jQuery object from the matching elements. The .parents() and .parent() methods are similar, except that the latter only travels a single level up the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the $() function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
```

```
</ul>
```

If we begin at item A, we can find its parents:

```
$('li.item-a').parent().css('background-color', 'red');
```

The result of this call is a red background for the level-2 list. Since we do not supply a selector expression, the parent element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

Example
Shows the parent of each element as (parent > child). Check the View Source to see the raw html.

```
$("*", document.body).each(function () {
   var parentTag = $(this).parent().get(0).tagName;
   $(this).prepend(document.createTextNode(parentTag + " > "));
  });
```

Example
Find the parent element of each paragraph with a class "selected".

```
$("p").parent(".selected").css("background", "yellow");
```

offsetParent()
Get the closest ancestor element that is positioned.

Given a jQuery object that represents a set of DOM elements, the `.offsetParent()` method allows us to search through the ancestors of these elements in the DOM tree and construct a new jQuery object wrapped around the closest positioned ancestor. An element is said to be positioned if it has a CSS position attribute of `relative`, `absolute`, or `fixed`. This information is useful for calculating offsets for performing animations and placing objects on the page.

Consider a page with a basic nested list on it, with a positioned element:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii" style="position: relative;">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at item A, we can find its positioned ancestor:

```
$('li.item-a').offsetParent().css('background-color', 'red');
```

This will change the color of list item II, which is positioned.

Example
Find the offsetParent of item "A."

```
$('li.item-a').offsetParent().css('background-color', 'red');
```

nextAll([selector])

Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

Arguments

selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.nextAll()` method allows us to search through the successors of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the elements which come after it:

```
$('li.third-item').nextAll().css('background-color', 'red');
```

The result of this call is a red background behind items 4 and 5. Since we do not supply a selector expression, these following elements are unequivocally included as part of the object. If we had supplied one, the elements would be tested for a match before they were included.

Example

Locate all the divs after the first and give them a class.

```
$("div:first").nextAll().addClass("after");
```

Example

Locate all the paragraphs after the second child in the body and give them a class.

```
$(":nth-child(1)").nextAll("p").addClass("after");
```

next([selector])

Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.

Arguments

selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.next()` method allows us to search through the immediately following sibling of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the immediately following sibling matches the selector, it remains in the newly constructed jQuery object; otherwise, it is excluded.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
```

```
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the element which comes just after it:

```
$('li.third-item').next().css('background-color', 'red');
```

The result of this call is a red background behind item 4. Since we do not supply a selector expression, this following element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

Example
Find the very next sibling of each disabled button and change its text "this button is disabled".

```
$("button[disabled]").next().text("this button is disabled");
```

Example
Find the very next sibling of each paragraph. Keep only the ones with a class "selected".

```
$("p").next(".selected").css("background", "yellow");
```

find(selector)
Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.find()` method allows us to search through the descendants of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.find()` and `.children()` methods are similar, except that the latter only travels a single level down the DOM tree.

The first signature for the `.find()`method accepts a selector expression of the same type that we can pass to the `$()` function. The elements will be filtered by testing whether they match this selector.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at item II, we can find list items within it:

```
$('li.item-ii').find('li').css('background-color', 'red');
```

The result of this call is a red background on items A, B, 1, 2, 3, and C. Even though item II matches the selector expression, it is not included in the results; only descendants are considered candidates for the match.

Unlike in the rest of the tree traversal methods, the selector expression is required in a call to `.find()`. If we need to retrieve all of the descendant elements, we can pass in the universal selector `'*'` to accomplish this.

Selector context is implemented with the `.find()method;` therefore, `$('li.item-ii').find('li')` is equivalent to `$('li', 'li.item-ii')`.

As of jQuery 1.6, we can also filter the selection with a given jQuery collection or element. With the same nested list as above, if we start with:

```
var $allListElements = $('li');
```

And then pass this jQuery object to find:

```
$('li.item-ii').find( $allListElements );
```

This will return a jQuery collection which contains only the list elements that are descendants of item II.

Similarly, an element may also be passed to find:

```
var item1 = $('li.item-1')[0];
$('li.item-ii').find( item1 ).css('background-color', 'red');
```

The result of this call would be a red background on item 1.

Example
Starts with all paragraphs and searches for descendant span elements, same as $("p span")

```
$("p").find("span").css('color','red');
```

Example
A selection using a jQuery collection of all span tags. Only spans within p tags are changed to red while others are left blue.

```
var $spans = $('span');
 $("p").find( $spans ).css('color','red');
```

Example
Add spans around each word then add a hover and italicize words with the letter **t**.

```
var newText = $("p").text().split(" ").join("</span> <span>");
 newText = "<span>" + newText + "</span>";

 $("p").html( newText )
  .find('span')
  .hover(function() {
   $(this).addClass("hilite");
  },
   function() { $(this).removeClass("hilite");
  })
 .end()
  .find(":contains('t')")
  .css({"font-style":"italic", "font-weight":"bolder"});
```

closest(selector)
Get the first element that matches the selector, beginning at the current element and progressing up through the DOM tree.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.closest()` method searches through these elements and their ancestors in the DOM tree and constructs a new jQuery object from the matching elements. The `.parents()` and `.closest()` methods are similar in that they both traverse up the DOM tree. The differences between the two, though subtle, are significant:

.closest()	.parents()
Begins with the current element	Begins with the parent element
Travels up the DOM tree until it finds a match for the supplied selector	Travels up the DOM tree to the document's root element, adding each ancestor element to a temporary collectio
The returned jQuery object contains zero or one element	The returned jQuery object contains zero, one, or multiple elements

```
<ul id="one" class="level-1">
 <li class="item-i">I</li>
 <li id="ii" class="item-ii">II
 <ul class="level-2">
  <li class="item-a">A</li>
  <li class="item-b">B
   <ul class="level-3">
    <li class="item-1">1</li>
    <li class="item-2">2</li>
    <li class="item-3">3</li>
   </ul>
  </li>
  <li class="item-c">C</li>
 </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

Suppose we perform a search for `<ul>` elements starting at item A:

```
$('li.item-a').closest('ul')
 .css('background-color', 'red');
```

This will change the color of the level-2 `<ul>`, since it is the first encountered when traveling up the DOM tree.

Suppose we search for an `<li>` element instead:

```
$('li.item-a').closest('li')
 .css('background-color', 'red');
```

This will change the color of list item A. The `.closest()` method begins its search *with the element itself* before progressing up the DOM tree, and stops when item A matches the selector.

We can pass in a DOM element as the context within which to search for the closest element.

```
var listItemII = document.getElementById('ii');
$('li.item-a').closest('ul', listItemII)
 .css('background-color', 'red');
$('li.item-a').closest('#one', listItemII)
 .css('background-color', 'green');
```

This will change the color of the level-2 `<ul>`, because it is both the first `<ul>` ancestor of list item A and a descendant of list item II. It will not change the color of the level-1 `<ul>`, however, because it is not a descendant of list item II.

Example

Show how event delegation can be done with closest. The closest list element toggles a yellow background when it or its descendent is clicked.

```
$( document ).bind("click", function( e ) {
  $( e.target ).closest("li").toggleClass("hilight");
 });
```

Example
Pass a jQuery object to closest. The closest list element toggles a yellow background when it or its descendent is clicked.

```
var $listElements = $("li").css("color", "blue");
 $( document ).bind("click", function( e ) {
  $( e.target ).closest( $listElements ).toggleClass("hilight");
 });
```

closest(selectors, [context])
Gets an array of all the elements and selectors matched against the current element up through the DOM tree.

Arguments
selectors - An array or string containing a selector expression to match elements against (can also be a jQuery object).
context - A DOM element within which a matching element may be found. If no context is passed in then the context of the jQuery set will be used instead.

This signature (only!) is deprecated as of jQuery 1.7. This method is primarily meant to be used internally or by plugin authors.

Example
Show how event delegation can be done with closest.

```
var close = $("li:first").closest(["ul", "body"]);
 $.each(close, function(i){
 $("li").eq(i).html( this.selector + ": " + this.elem.nodeName );
 });
```

children([selector])
Get the children of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.children()` method allows us to search through the children of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.children()` method differs from `.find()` in that `.children()` only travels a single level down the DOM tree while `.find()` can traverse down multiple levels to select descendant elements (grandchildren, etc.) as well. Note also that like most jQuery methods, `.children()` does not return text nodes; to get *all* children including text and comment nodes, use `.contents()`.

The `.children()` method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
```

```
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at the level-2 list, we can find its children:

```
$('ul.level-2').children().css('background-color', 'red');
```

The result of this call is a red background behind items A, B, and C. Since we do not supply a selector expression, all of the children are part of the returned jQuery object. If we had supplied one, only the matching items among these three would be included.

Example

Find all children of the clicked element.

```
$("#container").click(function (e) {
   $("*").removeClass("hilite");
   var $kids = $(e.target).children();
   var len = $kids.addClass("hilite").length;

   $("#results span:first").text(len);
   $("#results span:last").text(e.target.tagName);

   e.preventDefault();
   return false;
  });
```

Example

Find all children of each div.

```
$("div").children().css("border-bottom", "3px double red");
```

Example

Find all children with a class "selected" of each div.

```
$("div").children(".selected").css("color", "blue");
```

Utilities

jQuery.isNumeric(value)
Determines whether its argument is a number.

Arguments
value - The value to be tested.

The `$.isNumeric()` method checks whether its argument represents a numeric value. If so, it returns `true`. Otherwise it returns `false`. The argument can be of any type.

Example
Sample return values of $.isNumeric with various inputs.

```
$.isNumeric("-10"); // true
$.isNumeric(16);   // true
$.isNumeric(0xFF);  // true
$.isNumeric("0xFF"); // true
$.isNumeric("8e5"); // true (exponential notation string)
$.isNumeric(3.1415); // true
$.isNumeric(+10);  // true
$.isNumeric(0144);  // true (octal integer literal)
$.isNumeric("");   // false
$.isNumeric({});   // false (empty object)
$.isNumeric(NaN);  // false
$.isNumeric(null);  // false
$.isNumeric(true);  // false
$.isNumeric(Infinity); // false
$.isNumeric(undefined); // false
```

jQuery.now()
Return a number representing the current time.

The `$.now()` method is a shorthand for the number returned by the expression `(new Date).getTime()`.

jQuery.parseXML(data)
Parses a string into an XML document.

Arguments
data - a well-formed XML string to be parsed

`jQuery.parseXML` uses the native parsing function of the browser to create a valid XML Document. This document can then be passed to `jQuery` to create a typical jQuery object that can be traversed and manipulated.

Example
Create a jQuery object using an XML string and obtain the value of the title node.

```
var xml = "<rss version='2.0'><channel><title>RSS Title</title></channel></rss>",
  xmlDoc = $.parseXML( xml ),
  $xml = $( xmlDoc ),
  $title = $xml.find( "title" );

/* append "RSS Title" to #someElement */
$( "#someElement" ).append( $title.text() );

/* change the title to "XML Title" */
$title.text( "XML Title" );
```

```
/* append "XML Title" to #anotherElement */
$( "#anotherElement" ).append( $title.text() );
```

jQuery.type(obj)

Determine the internal JavaScript [[Class]] of an object.

Arguments

obj - Object to get the internal JavaScript [[Class]] of.

A number of techniques are used to determine the exact return value for an object. The [[Class]] is determined as follows:

- If the object is undefined or null, then "undefined" or "null" is returned accordingly.
- jQuery.type(undefined) === "undefined"
- jQuery.type() === "undefined"
- jQuery.type(window.notDefined) === "undefined"
- jQuery.type(null) === "null"
- If the object has an internal [[Class]] equivalent to one of the browser's built-in objects, the associated name is returned. (More details about this technique.)
- jQuery.type(true) === "boolean"
- jQuery.type(3) === "number"
- jQuery.type("test") === "string"
- jQuery.type(function(){}) === "function"
- jQuery.type([]) === "array"
- jQuery.type(new Date()) === "date"
- jQuery.type(/test/) === "regexp"
- Everything else returns "object" as its type.

Example

Find out if the parameter is a RegExp.

```
$("b").append( "" + jQuery.type(/test/) );
```

jQuery.isWindow(obj)

Determine whether the argument is a window.

Arguments

obj - Object to test whether or not it is a window.

This is used in a number of places in jQuery to determine if we're operating against a browser window (such as the current window or an iframe).

Example

Finds out if the parameter is a window.

```
$("b").append( "" + $.isWindow(window) );
```

jQuery.parseJSON(json)

Takes a well-formed JSON string and returns the resulting JavaScript object.

Arguments

json - The JSON string to parse.

Passing in a malformed JSON string may result in an exception being thrown. For example, the following are all malformed JSON strings:

- `{test: 1}` (test does not have double quotes around it).
- `{'test': 1}` ('test' is using single quotes instead of double quotes).

Additionally if you pass in nothing, an empty string, null, or undefined, 'null' will be returned from parseJSON. Where the browser provides a native implementation of `JSON.parse`, jQuery uses it to parse the string. For details on the JSON format, see http://json.org/.

Example

Parse a JSON string.

```
var obj = jQuery.parseJSON('{"name":"John"}');
alert( obj.name === "John" );
```

jQuery.proxy(function, context)

Takes a function and returns a new one that will always have a particular context.

Arguments

function - The function whose context will be changed.
context - The object to which the context (`this`) of the function should be set.

This method is most useful for attaching event handlers to an element where the context is pointing back to a different object. Additionally, jQuery makes sure that even if you bind the function returned from `jQuery.proxy()` it will still unbind the correct function if passed the original.

Be aware, however, that jQuery's event binding subsystem assigns a unique id to each event handling function in order to track it when it is used to specify the function to be unbound. The function represented by `jQuery.proxy()` is seen as a single function by the event subsystem, even when it is used to bind different contexts. To avoid unbinding the wrong handler, use a unique event namespace for binding and unbinding (e.g., `"click.myproxy1"`) rather than specifying the proxied function during unbinding.

Example

Change the context of functions bound to a click handler using the "function, context" signature. Unbind the first handler after first click.

```
var me = {
 type: "zombie",
 test: function(event) {
  // Without proxy, `this` would refer to the event target
  // use event.target to reference that element.
  var element = event.target;
  $(element).css("background-color", "red");

  // With proxy, `this` refers to the me object encapsulating
  // this function.
  $("#log").append( "Hello " + this.type + "<br>" );
  $("#test").unbind("click", this.test);
 }
};

var you = {
 type: "person",
 test: function(event) {
  $("#log").append( this.type + " " );
 }
};

// execute you.test() in the context of the `you` object
// no matter where it is called
// i.e. the `this` keyword will refer to `you`
var youClick = $.proxy( you.test, you );


// attach click handlers to #test
$("#test")
 // this === "zombie"; handler unbound after first click
 .click( $.proxy( me.test, me ) )
 // this === "person"
 .click( youClick )
 // this === "zombie"
 .click( $.proxy( you.test, me ) )
 // this === "<button> element"
 .click( you.test );
```

Example

Enforce the context of the function using the "context, function name" signature. Unbind the handler after first click.

```
var obj = {
  name: "John",
  test: function() {
   $("#log").append( this.name );
   $("#test").unbind("click", obj.test);
  }
 };

 $("#test").click( jQuery.proxy( obj, "test" ) );
```

jQuery.contains(container, contained)

Check to see if a DOM element is within another DOM element.

Arguments

container - The DOM element that may contain the other element.
contained - The DOM element that may be contained by the other element.

Note: The first argument *must* be a DOM element, not a jQuery object or plain JavaScript object.

Example

Check if an element is inside another. Text and comment nodes are not supported.

```
jQuery.contains(document.documentElement, document.body); // true
jQuery.contains(document.body, document.documentElement); // false
```

jQuery.noop()

An empty function.

You can use this empty function when you wish to pass around a function that will do nothing.

This is useful for plugin authors who offer optional callbacks; in the case that no callback is given, something like `jQuery.noop` could execute.

jQuery.globalEval(code)

Execute some JavaScript code globally.

Arguments

code - The JavaScript code to execute.

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).

Example

Execute a script in the global context.

```
function test(){
  jQuery.globalEval("var newVar = true;")
}
test();
// newVar === true
```

jQuery.isXMLDoc(node)

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

Arguments

node - The DOM node that will be checked to see if it's in an XML document.

Example

Check an object to see if it's in an XML document.

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

jQuery.removeData(element, [name])

Remove a previously-stored piece of data.

Arguments

element - A DOM element from which to remove data.
name - A string naming the piece of data to remove.

Note: This is a low-level method, you should probably use `.removeData()` instead.

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

Example

Set a data store for 2 names then remove one of them.

```
var div = $("div")[0];
$("span:eq(0)").text("" + $("div").data("test1"));
jQuery.data(div, "test1", "VALUE-1");
jQuery.data(div, "test2", "VALUE-2");
$("span:eq(1)").text("" + jQuery.data(div, "test1"));
jQuery.removeData(div, "test1");
$("span:eq(2)").text("" + jQuery.data(div, "test1"));
$("span:eq(3)").text("" + jQuery.data(div, "test2"));
```

jQuery.data(element, key, value)

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

Arguments

element - The DOM element to associate with the data.
key - A string naming the piece of data to set.
value - The new data value.

Note: This is a low-level method; a more convenient `.data()` is also available.

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');
```

Note: 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.

Example

Store then retrieve a value from the div element.

```
var div = $("div")[0];
  jQuery.data(div, "test", { first: 16, last: "pizza!" });
  $("span:first").text(jQuery.data(div, "test").first);
  $("span:last").text(jQuery.data(div, "test").last);
```

jQuery.data(element, key)

Returns value at named data store for the element, as set by `jQuery.data(element, name, value)`, or the full data store for the element.

Arguments
element - The DOM element to query for the data.
key - Name of the data stored.

Note: This is a low-level method; a more convenient `.data()` is also available.

Regarding HTML5 data-* attributes: This low-level method does NOT retrieve the `data-*` attributes unless the more convenient `.data()` method has already retrieved them.

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 from memory leaks. We can retrieve several distinct values for a single element one at a time, or as a set:

```
alert(jQuery.data( document.body, 'foo' ));
alert(jQuery.data( document.body ));
```

The above lines alert the data values that were set on the `body` element. If nothing was set on that element, an empty string is returned.

Calling `jQuery.data(element)` retrieves all of the element's associated values as a JavaScript object. Note that jQuery itself uses this method to store data for internal use, such as event handlers, so do not assume that it contains only data that your own code has stored.

Note: 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.

Example
Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value, div = $("div")[0];

 switch ($("button").index(this)) {
  case 0 :
   value = jQuery.data(div, "blah");
   break;
  case 1 :
   jQuery.data(div, "blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   jQuery.data(div, "blah", 86);
   value = "Stored!";
   break;
  case 3 :
   jQuery.removeData(div, "blah");
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

jQuery.dequeue(element, [queueName])
Execute the next function on the queue for the matched element.

Arguments
element - A DOM element from which to remove and execute a queued function.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Note: This is a low-level method, you should probably use `.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.

Example
Use jQuery.dequeue() to end a custom queue function which allows the queue to keep going.

```
$("button").click(function () {
   $("div").animate({left:'+=200px'}, 2000);
   $("div").animate({top:'0px'}, 600);
   $("div").queue(function () {
    $(this).toggleClass("red");
     $.dequeue( this );
       });
   $("div").animate({left:'10px', top:'30px'}, 700);
  });
```

jQuery.queue(element, [queueName])
Show the queue of functions to be executed on the matched element.

Arguments
element - A DOM element to inspect for an attached queue.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Note: This is a low-level method, you should probably use `.queue()` instead.

Example
Show the length of the queue.

```
$("#show").click(function () {
   var n = jQuery.queue( $("div")[0], "fx" );
   $("span").text("Queue length is: " + n.length);
  });
  function runIt() {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   $("div").slideToggle(1000);
   $("div").slideToggle("fast");
   $("div").animate({left:'-=200'},1500);
   $("div").hide("slow");
   $("div").show(1200);
   $("div").slideUp("normal", runIt);
  }
  runIt();
```

jQuery.queue(element, queueName, newQueue)
Manipulate the queue of functions to be executed on the matched element.

Arguments
element - A DOM element where the array of queued functions is attached.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.
newQueue - An array of functions to replace the current queue contents.

Note: This is a low-level method, you should probably use `.queue()` instead.

Every element can have one or more queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution.

The `jQuery.queue()` method allows us to directly manipulate this queue of functions. Calling `jQuery.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue.

Note that when adding a function with `jQuery.queue()`, we should ensure that `jQuery.dequeue()` is eventually called so that the next function in line executes.

Example

Queue a custom function.

```
$(document.body).click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).addClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").animate({left:'-=200'},500);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).removeClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").slideUp();
  });
```

Example
Set a queue array to delete the queue.

```
$("#start").click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},5000);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).addClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").animate({left:'-=200'},1500);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).removeClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").slideUp();
  });
  $("#stop").click(function () {
   jQuery.queue( $("div")[0], "fx", [] );
   $("div").stop();
  });
```

clearQueue([queueName])
Remove from the queue all items that have not yet been run.

Arguments
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

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.

Example
Empty the queue.

```
$("#start").click(function () {

 var myDiv = $("div");
 myDiv.show("slow");
 myDiv.animate({left:'+=200'},5000);
 myDiv.queue(function () {
  var _this = $(this);
  _this.addClass("newcolor");
  _this.dequeue();
```

```
 });

 myDiv.animate({left:'-=200'},1500);
 myDiv.queue(function () {
  var _this = $(this);
  _this.removeClass("newcolor");
  _this.dequeue();
 });
 myDiv.slideUp();

});

$("#stop").click(function () {
 var myDiv = $("div");
 myDiv.clearQueue();
 myDiv.stop();
});
```

jQuery.isEmptyObject(object)

Check to see if an object is empty (contains no properties).

Arguments

object - The object that will be checked to see if it's empty.

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()

Example

Check an object to see if it's empty.

```
jQuery.isEmptyObject({}) // true
jQuery.isEmptyObject({ foo: "bar" }) // false
```

jQuery.isPlainObject(object)

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

Arguments

object - The object that will be checked to see if it's a plain 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.

An example of this is a test against `document.location` using `$.isPlainObject()` as follows:

```
console.log($.isPlainObject(document.location));
```

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. Some basic example of use-cases that do function correctly cross-browser can be found below.

Example

Check an object to see if it's a plain object.

```
jQuery.isPlainObject({}) // true
jQuery.isPlainObject("test") // false
```

dequeue([queueName])

Execute the next function on the queue for the matched elements.

Arguments
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

When `.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 `.dequeue()` to be called, so that the sequence can continue.

Example
Use dequeue to end a custom queue function which allows the queue to keep going.

```
$("button").click(function () {
 $("div").animate({left:'+=200px'}, 2000);
 $("div").animate({top:'0px'}, 600);
 $("div").queue(function () {
  $(this).toggleClass("red");
  $(this).dequeue();
 });
 $("div").animate({left:'10px', top:'30px'}, 700);
});
```

queue([queueName])
Show the queue of functions to be executed on the matched elements.

Arguments
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Example
Show the length of the queue.

```
var div = $("div");

function runIt() {
 div.show("slow");
 div.animate({left:'+=200'},2000);
 div.slideToggle(1000);
 div.slideToggle("fast");
 div.animate({left:'-=200'},1500);
 div.hide("slow");
 div.show(1200);
 div.slideUp("normal", runIt);
}

function showIt() {
 var n = div.queue("fx");
 $("span").text( n.length );
 setTimeout(showIt, 100);
}

runIt();
showIt();
```

queue([queueName], newQueue)
Manipulate the queue of functions to be executed on the matched elements.

Arguments
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.
newQueue - An array of functions to replace the current queue contents.

Every element can have one to many queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution. The typical example of this is calling

multiple animation methods on an element. For example:

```
$('#foo').slideUp().fadeIn();
```

When this statement is executed, the element begins its sliding animation immediately, but the fading transition is placed on the `fx` queue to be called only once the sliding transition is complete.

The `.queue()` method allows us to directly manipulate this queue of functions. Calling `.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue.

This feature is similar to providing a callback function with an animation method, but does not require the callback to be given at the time the animation is performed.

```
$('#foo').slideUp();
$('#foo').queue(function() {
 alert('Animation complete.');
 $(this).dequeue();
});
```

This is equivalent to:

```
$('#foo').slideUp(function() {
 alert('Animation complete.');
});
```

Note that when adding a function with `.queue()`, we should ensure that `.dequeue()` is eventually called so that the next function in line executes.

As of jQuery 1.4, the function that's called is passed another function as the first argument. When called, this automatically dequeues the next item and keeps the queue moving. We use it as follows:

```
$("#test").queue(function(next) {
  // Do some stuff...
  next();
});
```

Example

Queue a custom function.

```
$(document.body).click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
   });
   $("div").animate({left:'-=200'},500);
   $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
   });
   $("div").slideUp();
  });
```

Example

Set a queue array to delete the queue.

```
$("#start").click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},5000);
   $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
   });
```

```
  $("div").animate({left:'-=200'},1500);
  $("div").queue(function () {
   $(this).removeClass("newcolor");
   $(this).dequeue();
  });
  $("div").slideUp();
 });
 $("#stop").click(function () {
  $("div").queue("fx", []);
  $("div").stop();
 });
```

jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. **We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.**

The `$.browser` property provides information about the web browser that is accessing the page, as reported by the browser itself. It contains flags for each of the four most prevalent browser classes (Internet Explorer, Mozilla, Webkit, and Opera) as well as version information.

Available flags are:

- webkit (as of jQuery 1.4)
- safari (deprecated)
- opera
- msie
- mozilla

This property is available immediately. It is therefore safe to use it to determine whether or not to call `$(document).ready()`.The `$.browser` property is deprecated in jQuery 1.3, and its functionality may be moved to a team-supported plugin in a future release of jQuery.

Because `$.browser` uses `navigator.userAgent` to determine the platform, it is vulnerable to spoofing by the user or misrepresentation by the browser itself. It is always best to avoid browser-specific code entirely where possible. The `$.support` property is available for detection of support for particular features rather than relying on `$.browser`.

Example

Show the browser info.

```
jQuery.each(jQuery.browser, function(i, val) {
   $("<div>" + i + " : <span>" + val + "</span>")
        .appendTo( document.body );
 });
```

Example

Returns true if the current useragent is some version of Microsoft's Internet Explorer.

```
$.browser.msie;
```

Example

Alerts "this is WebKit!" only for WebKit browsers

```
if ($.browser.webkit) {
  alert( "this is webkit!" );
 }
```

Example

Alerts "Do stuff for Firefox 3" only for Firefox 3 browsers.

```
var ua = $.browser;
 if ( ua.mozilla && ua.version.slice(0,3) == "1.9" ) {
  alert( "Do stuff for firefox 3" );
 }
```

Example
Set a CSS property that's specific to a particular browser.

```
if ( $.browser.msie ) {
  $("#div ul li").css( "display","inline" );
 } else {
  $("#div ul li").css( "display","inline-table" );
 }
```

jQuery.browser.version
The version number of the rendering engine for the user's browser.

Here are some typical results:
- Internet Explorer: 6.0, 7.0, 8.0
- Mozilla/Firefox/Flock/Camino: 1.7.12, 1.8.1.3, 1.9
- Opera: 10.06, 11.01
- Safari/Webkit: 312.8, 418.9

Note that IE8 claims to be 7 in Compatibility View.

Example
Returns the version number of the rendering engine used by the user's current browser. For example, FireFox 4 returns 2.0 (the version of the Gecko rendering engine it utilizes).

```
$("p").html( "The version number of the rendering engine your browser uses is: <span>" +
        $.browser.version + "</span>" );
```

Example
Alerts the version of IE's rendering engine that is being used:

```
if ( $.browser.msie ) {
 alert( $.browser.version );
}
```

Example
Often you only care about the "major number," the whole number, which you can get by using JavaScript's built-in `parseInt()` function:

```
if ( $.browser.msie ) {
 alert( parseInt($.browser.version, 10) );
}
```

jQuery.trim(str)
Remove the whitespace from the beginning and end of a string.

Arguments
str - The string to trim.

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.

Example
Remove the two white spaces at the start and at the end of the string.

```
var str = "     lots of spaces before and after     ";
 $("#original").html("Original String: '" + str + "'");
 $("#trimmed").html("$.trim()'ed: '" + $.trim(str) + "'");
```

Example
Remove the two white spaces at the start and at the end of the string.

```
$.trim("  hello, how are you?  ");
```

jQuery.isFunction(obj)
Determine if the argument passed is a Javascript function object.

Arguments
obj - Object to test whether or not it is a function.

Note: As of jQuery 1.3, functions provided by the browser like `alert()` and DOM element methods like `getAttribute()` are not guaranteed to be detected as functions in browsers such as Internet Explorer.

Example
Test a few parameter examples.

```
function stub() {
  }
  var objs = [
     function () {},
     { x:15, y:20 },
     null,
     stub,
     "function"
    ];

  jQuery.each(objs, function (i) {
   var isFunc = jQuery.isFunction(objs[i]);
   $("span").eq(i).text(isFunc);
  });
```

Example
Finds out if the parameter is a function.

```
$.isFunction(function(){});
```

jQuery.isArray(obj)
Determine whether the argument is an array.

Arguments
obj - Object to test whether or not it is an array.

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

Example
Finds out if the parameter is an array.

```
$("b").append( "" + $.isArray([]) );
```

jQuery.unique(array)
Sorts an array of DOM elements, in place, with the duplicates removed. Note that this only works on arrays of DOM elements, not strings or numbers.

Arguments
array - The Array of DOM elements.

The `$.unique()` function searches through an array of objects, sorting the array, and removing any duplicate nodes. This function only works on plain JavaScript arrays of DOM elements, and is chiefly used internally by jQuery.

As of jQuery 1.4 the results will always be returned in document order.

Example
Removes any duplicate elements from the array of divs.

```
var divs = $("div").get(); // unique() must take a native array

  // add 3 elements of class dup too (they are divs)
  divs = divs.concat($(".dup").get());
  $("div:eq(1)").text("Pre-unique there are " + divs.length + " elements.");

  divs = jQuery.unique(divs);
  $("div:eq(2)").text("Post-unique there are " + divs.length + " elements.")
         .css("color", "red");
```

jQuery.merge(first, second)

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

Arguments

first - The first array to merge, the elements of second added.
second - The second array to merge into the first, unaltered.

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 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:

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

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

Prior to jQuery 1.4, the arguments should be true Javascript Array objects; use `$.makeArray` if they are not.

Example

Merges two arrays, altering the first argument.

```
$.merge( [0,1,2], [2,3,4] )
```

Example

Merges two arrays, altering the first argument.

```
$.merge( [3,2,1], [4,3,2] )
```

Example

Merges two arrays, but uses a copy, so the original isn't altered.

```
var first = ['a','b','c'];
var second = ['d','e','f'];
$.merge( $.merge([],first), second);
```

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

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

Arguments

value - The value to search for.
array - An array through which to search.
fromIndex - The index of the array at which to begin the search. The default is 0, which will search the whole array.

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), 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.

Example

Report the index of some elements in the array.

```
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));
```

jQuery.map(array, callback(elementOfArray, indexInArray))

Translate all items in an array or object to new array of items.

Arguments

array - The Array to translate.
callback(elementOfArray, indexInArray) - The function to process each item against. The first argument to the function is the array item, the second argument is the index in array The function can return any value. Within the function, `this` refers to the global (window) object.

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 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 - must be converted to actual arrays before being passed to `$.map()`. The jQuery library provides $.makeArray() for such conversions.

```
// The following object masquerades as an array.
var fakeArray = {"length": 1, 0: "Addy", 1: "Subtracty"};

// Therefore, convert it to a real array
var realArray = $.makeArray( fakeArray )

// Now it can be used reliably with $.map()
$.map( realArray, function(val, i) {
 // do something
});
```

The translation function that is provided to this method is called for each top-level element in the array or object and is passed two arguments: The element's value and its index or key within the array or object.

The function can return:

- the translated value, which will be mapped to the resulting array
- `null`, to remove the item
- an array of values, which will be flattened into the full array

Example

A couple examples of using .map()

```
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(", "));
```

Example

Map the original array to a new one and add 4 to each value.

```
$.map( [0,1,2], function(n){
```

```
  return n + 4;
 });
```

Example
Maps the original array to a new one and adds 1 to each value if it is bigger then zero, otherwise it's removed.

```
$.map( [0,1,2], function(n){
  return n > 0 ? n + 1 : null;
 });
```

Example
Map the original array to a new one; each element is added with its original value and the value plus one.

```
$.map( [0,1,2], function(n){
  return [ n, n + 1 ];
 });
```

Example
Map the original object to a new array and double each value.

```
var dimensions = { width: 10, height: 15, length: 20 };
dimensions = $.map( dimensions, function( value, index ) {
 return value * 2;
});
```

Example
Map an object's keys to an array.

```
var dimensions = { width: 10, height: 15, length: 20 },
  keys = $.map( dimensions, function( value, index ) {
   return index;
  });
```

Example
Maps the original array to a new one; each element is squared.

```
$.map( [0,1,2,3], function (a) {
 return a * a;
});
```

Example
Remove items by returning `null` from the function. This removes any numbers less than 50, and the rest are decreased by 45.

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

Example
Augmenting the resulting array by returning an array inside the function.

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

jQuery.makeArray(obj)
Convert an array-like object into a true JavaScript array.

Arguments
obj - Any object to turn into a native 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.

Example
Turn a collection of HTMLElements into an Array of them.

```
var elems = document.getElementsByTagName("div"); // returns a nodeList
  var arr = jQuery.makeArray(elems);
  arr.reverse(); // use an Array method on list of dom elements
  $(arr).appendTo(document.body);
```

Example
Turn a jQuery object into an array

```
var obj = $('li');
  var arr = $.makeArray(obj);
```

jQuery.grep(array, function(elementOfArray, indexInArray), [invert])
Finds the elements of an array which satisfy a filter function. The original array is not affected.

Arguments
array - The array to search through.
function(elementOfArray, indexInArray) - The function to process each item against. The first argument to the function is the item, and the second argument is the index. The function should return a Boolean value. `this` will be the global window object.
invert - If "invert" is false, or not provided, then the function returns an array consisting of all elements for which "callback" returns true. If "invert" is true, then the function returns an array consisting of all elements for which "callback" returns false.

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.

Example
Filters the original array of numbers leaving that are not 5 and have an index greater than 4. Then it removes all 9s.

```
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(", "));
```

Example
Filter an array of numbers to include only numbers bigger then zero.

```
$.grep( [0,1,2], function(n,i){
  return n > 0;
 });
```

Example
Filter an array of numbers to include numbers that are not bigger than zero.

```
$.grep( [0,1,2], function(n,i){
  return n > 0;
```

```
},true);
```

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

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

Arguments

target - An object that will receive the new properties if additional objects are passed in or that will extend the jQuery namespace if it is the sole argument.
object1 - An object containing additional properties to merge in.
objectN - Additional objects containing properties to merge in.

When we supply two or more objects to `$.extend()`, 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. 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, we want to preserve both of the original objects, we can do so by passing an empty object as the target:

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

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 example below by examining the value of banana. However, by passing `true` for the first function argument, objects will be recursively merged. (Passing `false` for the first argument is not supported.)

Undefined properties are not copied. However, properties inherited from the object's prototype *will* be copied over. Properties that are an object constructed via `new MyCustomObject(args)`, or built-in JavaScript types such as Date or RegExp, are not re-constructed and will appear as plain Objects in the resulting object or array.

On a `deep` extend, Object and Array are extended, but object wrappers on primitive types such as String, Boolean, and Number are not.

For needs that fall outside of this behavior, write a custom extend method instead.

Example

Merge two objects, modifying the first.

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

/* merge object2 into object1 */
$.extend(object1, object2);

var printObj = typeof JSON != "undefined" ? JSON.stringify : function(obj) {
 var arr = [];
 $.each(obj, function(key, val) {
  var next = key + ": ";
  next += $.isPlainObject(val) ? printObj(val) : val;
  arr.push( next );
 });
 return "{ " + arr.join(", ") + " }";
};

$("#log").append( printObj(object1) );
```

Example
Merge two objects recursively, modifying the first.

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

/* merge object2 into object1, recursively */
$.extend(true, object1, object2);

var printObj = typeof JSON != "undefined" ? JSON.stringify : function(obj) {
 var arr = [];
 $.each(obj, function(key, val) {
  var next = key + ": ";
  next += $.isPlainObject(val) ? printObj(val) : val;
  arr.push( next );
 });
 return "{ " + arr.join(", ") + " }";
};

$("#log").append( printObj(object1) );
```

Example
Merge defaults and options, without modifying the defaults. This is a common plugin development pattern.

```
var defaults = { validate: false, limit: 5, name: "foo" };
var options = { validate: true, name: "bar" };

/* merge defaults and options, without modifying defaults */
var settings = $.extend({}, defaults, options);

var printObj = typeof JSON != "undefined" ? JSON.stringify : function(obj) {
 var arr = [];
 $.each(obj, function(key, val) {
  var next = key + ": ";
  next += $.isPlainObject(val) ? printObj(val) : val;
  arr.push( next );
 });
 return "{ " + arr.join(", ") + " }";
};


$("#log").append( "<div><b>defaults -- </b>" + printObj(defaults) + "</div>" );
$("#log").append( "<div><b>options -- </b>" + printObj(options) + "</div>" );
$("#log").append( "<div><b>settings -- </b>" + printObj(settings) + "</div>" );
```

jQuery.each(collection, callback(indexInArray, valueOfElement))
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.

Arguments
collection - The object or array to iterate over.
callback(indexInArray, valueOfElement) - The function that will be executed on every object.

The `$.each()` function is not the same as $(selector).each(), which is used to iterate, exclusively, over a jQuery object. The `$.each()` function can be used to iterate over any collection, whether it is 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. (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: 521: 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(key, value) {
 alert(key + ': ' + value);
});
```

Once again, this produces two messages:

flammable: inflammableduh: no duh

We can break the `$.each()` loop at a particular iteration by making the callback function return `false`. Returning *non-false* is the same as a `continue` statement in a for loop; it will skip immediately to the next iteration.

Example
Iterates through the array displaying each number as both a word and numeral

```
var arr = [ "one", "two", "three", "four", "five" ];
  var obj = { one:1, two:2, three:3, four:4, five:5 };

  jQuery.each(arr, function() {
   $("#" + this).text("Mine is " + this + ".");
    return (this != "three"); // will stop running after "three"
  });

  jQuery.each(obj, function(i, val) {
   $("#" + i).append(document.createTextNode(" - " + val));
  });
```

Example
Iterates over items in an array, accessing both the current item and its index.

```
$.each( ['a','b','c'], function(i, l){
  alert( "Index #" + i + ": " + l );
 });
```

Example
Iterates over the properties in an object, accessing both the current item and its key.

```
$.each( { name: "John", lang: "JS" }, function(k, v){
  alert( "Key: " + k + ", Value: " + v );
 });
```

jQuery.boxModel
Deprecated in jQuery 1.3 (see jQuery.support). States if the current page, in the user's browser, is being rendered using the W3C CSS Box Model.

Example
Returns the box model for the iframe.

```
$("p").html("The box model for this iframe is: <span>" +
        jQuery.boxModel + "</span>");
```

Example
Returns false if the page is in Quirks Mode in Internet Explorer

```
$.boxModel
```

jQuery.support
A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Rather than using `$.browser` to detect the current user agent and alter the page presentation based on which browser is running, it is a good practice to perform **feature detection**. This means that prior to executing code which relies on a browser feature, we test to ensure that the feature works properly. To make this process simpler, jQuery performs many such tests and makes the results available to us as properties of the `jQuery.support` object.

The values of all the support properties are determined using feature detection (and do not use any form of browser sniffing).

Following are a few resources that explain how feature detection works:

- http://peter.michaux.ca/articles/feature-detection-state-of-the-art-browser-scripting
- http://www.jibbering.com/faq/faq_notes/not_browser_detect.html
- http://yura.thinkweb2.com/cft/

While jQuery includes a number of properties, developers should feel free to add their own as their needs dictate. Many of the `jQuery.support` properties are rather low-level, so they are most useful for plugin and jQuery core development, rather than general day-to-day development. Since jQuery requires these tests internally, they must be performed on *every* page load; for that reason this list is kept short and limited to features needed by jQuery itself.

The tests included in `jQuery.support` are as follows:

- `ajax` is equal to true if a browser is able to create an `XMLHttpRequest` object.
- `boxModel` is equal to true if the page is rendering according to the W3C CSS Box Model (is currently false in IE 6 and 7 when they are in Quirks Mode). This property is null until document ready occurs.
- `changeBubbles` is equal to true if the change event bubbles up the DOM tree, as required by the W3C DOM event model. (It is currently false in IE, and jQuery simulates bubbling).
- `checkClone` is equal to true if a browser correctly clones the checked state of radio buttons or checkboxes in document fragments.
- `checkOn` is equal to true if the value of a checkbox defaults to "on" when no value is specified.
- `cors` is equal to true if a browser can create an `XMLHttpRequest` object and if that `XMLHttpRequest` object has a `withCredentials` property. To enable cross-domain requests in environments that do not support cors yet but do allow cross-domain XHR requests (windows gadget, etc), set `$.support.cors = true;`.CORS WD
- `cssFloat` is equal to true if the name of the property containing the CSS float value is .cssFloat, as defined in the CSS Spec. (It is currently false in IE, it uses styleFloat instead).
- `hrefNormalized` is equal to true if the `.getAttribute()` method retrieves the `href` attribute of elements unchanged, rather than normalizing it to a fully-qualified URL. (It is currently false in IE, the URLs are normalized). DOM l3 spec
- `htmlSerialize` is equal to true if the browser is able to serialize/insert `<link>` elements using the `.innerHTML` property of elements. (is currently false in IE). HTML5 WD
- `leadingWhitespace` is equal to true if the browser inserts content with .innerHTML exactly as provided-specifically, if leading whitespace characters are preserved. (It is currently false in IE 6-8). HTML5 WD
- `noCloneChecked` is equal to true if cloned DOM elements copy over the state of the `.checked` expando. (It is currently false in IE). (Added in jQuery 1.5.1)
- `noCloneEvent` is equal to true if cloned DOM elements are created without event handlers (that is, if the event handlers on the source element are not cloned). (It is currently false in IE). DOM l2 spec
- `opacity` is equal to true if a browser can properly interpret the opacity style property. (It is currently false in IE, it uses alpha filters instead). CSS3 spec
- `optDisabled` is equal to true if option elements within disabled select elements are not automatically marked as disabled. HTML5 WD
- `optSelected` is equal to true if an `<option>` element that is selected by default has a working `selected` property. HTML5 WD
- `scriptEval()` is equal to true if inline scripts are automatically evaluated and executed when inserted into the document using standard DOM manipulation methods such as `.appendChild()` and `.createTextNode()`. (It is currently false in IE, it uses `.text` to insert executable scripts).

Note: No longer supported; removed in jQuery 1.6. Prior to jQuery 1.5.1, the `scriptEval()` method was the static `scriptEval` property. The change to a method allowed the test to be deferred until first use to prevent content security policy inline-script violations. HTML5 WD

• `style` is equal to true if inline styles for an element can be accessed through the DOM attribute called style, as required by the DOM Level 2 specification. In this case, `.getAttribute('style')` can retrieve this value; in Internet Explorer, `.cssText` is used for this purpose. DOM l2 Style spec

• `submitBubbles` is equal to true if the submit event bubbles up the DOM tree, as required by the W3C DOM event model. (It is currently false in IE, and jQuery simulates bubbling).

• `tbody` is equal to true if an empty `<table>` element can exist without a `<tbody>` element. According to the HTML specification, this sub-element is optional, so the property should be true in a fully-compliant browser. If false, we must account for the possibility of the browser injecting `<tbody>` tags implicitly. (It is currently false in IE, which automatically inserts `tbody` if it is not present in a string assigned to `innerHTML`). HTML5 spec

Example

Returns the box model for the iframe.

```
$("p").html("This frame uses the W3C box model: <span>" +
        jQuery.support.boxModel + "</span>");
```

Example

Returns false if the page is in QuirksMode in Internet Explorer

```
jQuery.support.boxModel
```

Version

Version 1.0

toggle(handler(eventObject), handler(eventObject), [handler(eventObject)])

Bind two or more handlers to the matched elements, to be executed on alternate clicks.

Arguments

handler(eventObject) - A function to execute every even time the element is clicked.
handler(eventObject) - A function to execute every odd time the element is clicked.
handler(eventObject) - Additional handlers to cycle through after clicks.

Note: jQuery also provides an animation method named .toggle() that toggles the visibility of elements. Whether the animation or the event method is fired depends on the set of arguments passed.

The `.toggle()` method binds a handler for the `click` event, so the rules outlined for the triggering of `click` apply here as well.

```
For example, consider the HTML:
<div id="target">
 Click here
</div>
```

Event handlers can then be bound to the `<div>`:

```
$('#target').toggle(function() {
 alert('First handler for .toggle() called.');
}, function() {
 alert('Second handler for .toggle() called.');
});
```

As the element is clicked repeatedly, the messages alternate:

First handler for .toggle() called.Second handler for .toggle() called.First handler for .toggle() called.Second handler for .toggle() called.First handler for .toggle() called.

If more than two handlers are provided, `.toggle()` will cycle among all of them. For example, if there are three handlers, then the first handler will be called on the first click, the fourth click, the seventh click, and so on.

The `.toggle()` method is provided for convenience. It is relatively straightforward to implement the same behavior by hand, and this can be necessary if the assumptions built into `.toggle()` prove limiting. For example, `.toggle()` is not guaranteed to work correctly if applied twice to the same element. Since `.toggle()` internally uses a `click` handler to do its work, we must unbind `click` to remove a behavior attached with `.toggle()`, so other `click` handlers can be caught in the crossfire. The implementation also calls `.preventDefault()` on the event, so links will not be followed and buttons will not be clicked if `.toggle()` has been called on the element.

Example

Click to toggle highlight on the list item.

```
$("li").toggle(
   function () {
    $(this).css({"list-style-type":"disc", "color":"blue"});
   },
   function () {
    $(this).css({"list-style-type":"disc", "color":"red"});
   },
   function () {
    $(this).css({"list-style-type":"", "color":""});
   }
  );
```

Example

To toggle a style on table cells:

```
$("td").toggle(
 function () {
  $(this).addClass("selected");
 },
 function () {
  $(this).removeClass("selected");
 }
);
```

event.stopPropagation()

Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event.

We can use `event.isPropagationStopped()` to determine if this method was ever called (on that event object).

This method works for custom events triggered with trigger(), as well.

Note that this will not prevent other handlers *on the same element* from running.

Example

Kill the bubbling on the click event.

```
$("p").click(function(event){
 event.stopPropagation();
 // do something
});
```

event.preventDefault()

If this method is called, the default action of the event will not be triggered.

For example, clicked anchors will not take the browser to a new URL. We can use `event.isDefaultPrevented()` to determine if this method has been called by an event handler that was triggered by this event.

Example

Cancel the default action (navigation) of the click.

```
$("a").click(function(event) {
 event.preventDefault();
 $('<div/>')
  .append('default ' + event.type + ' prevented')
  .appendTo('#log');
});
```

event.target

The DOM element that initiated the event.

The `target` property can be the element that registered for the event or a descendant of it. It is often useful to compare `event.target` to `this` in order to determine if the event is being handled due to event bubbling. This property is very useful in event delegation, when events bubble.

Example

Display the tag's name on click

```
$("body").click(function(event) {
 $("#log").html("clicked: " + event.target.nodeName);
});
```

Example

Implements a simple event delegation: The click handler is added to an unordered list, and the children of its li children are hidden. Clicking one of the

li children toggles (see toggle()) their children.

```
function handler(event) {
 var $target = $(event.target);
 if( $target.is("li") ) {
  $target.children().toggle();
 }
}
$("ul").click(handler).find("ul").hide();
```

event.type
Describes the nature of the event.

Example
On all anchor clicks, alert the event type.

```
$("a").click(function(event) {
 alert(event.type); // "click"
});
```

each(function(index, Element))
Iterate over a jQuery object, executing a function for each matched element.

Arguments
function(index, Element) - A function to execute for each matched element.

The `.each()` method is designed to make DOM looping constructs concise and less error-prone. When called it iterates over the DOM elements that are part of the jQuery 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 keyword `this` refers to the element.

Suppose we had a simple unordered list on the page:

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

We can select the list items and iterate across them:

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

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

0: foo1: bar

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

Example
Iterates over three divs and sets their color property.

```
$(document.body).click(function () {
   $("div").each(function (i) {
    if (this.style.color != "blue") {
     this.style.color = "blue";
    } else {
     this.style.color = "";
```

```
    }
   });
  });
```

Example
If you want to have the jQuery object instead of the regular DOM element, use the $(this) function, for example:

```
$("span").click(function () {
   $("li").each(function(){
    $(this).toggleClass("example");
   });
  });
```

Example
You can use 'return' to break out of each() loops early.

```
$("button").click(function () {
   $("div").each(function (index, domEle) {
    // domEle == this
    $(domEle).css("backgroundColor", "yellow");
    if ($(this).is("#stop")) {
     $("span").text("Stopped at div index #" + index);
     return false;
    }
   });
  });
```

pushStack(elements)
Add a collection of DOM elements onto the jQuery stack.

Arguments
elements - An array of elements to push onto the stack and make into a new jQuery object.

Example
Add some elements onto the jQuery stack, then pop back off again.

```
jQuery([])
  .pushStack( document.getElementsByTagName("div") )
    .remove()
  .end();
```

keydown(handler(eventObject))
Bind an event handler to the "keydown" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('keydown', handler)` in the first and second variations, and `.trigger('keydown')` in the third.

The `keydown` event is sent to an element when the user first presses a key on the keyboard. It can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$('#target').keydown(function() {
 alert('Handler for .keydown() called.');
});
```

Now when the insertion point is inside the field, pressing a key displays the alert:

Handler for .keydown() called.

To trigger the event manually, apply `.keydown()` without an argument:

```
$('#other').click(function() {
 $('#target').keydown();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which key was pressed, examine the event object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the key code. This code corresponds to a key on the keyboard, including codes for special keys such as arrows. For catching actual text entry, `.keypress()` may be a better choice.

Example

Show the event object for the keydown handler when a key is pressed in the input.

```
var xTriggered = 0;
$('#target').keydown(function(event) {
 if (event.which == 13) {
   event.preventDefault();
  }
  xTriggered++;
  var msg = 'Handler for .keydown() called ' + xTriggered + ' time(s).';
 $.print(msg, 'html');
 $.print(event);
});

$('#other').click(function() {
 $('#target').keydown();
});
```

index()

Search for a given element from among the matched elements.

Return Values

If no argument is passed to the `.index()` method, the return value is an integer indicating the position of the first element within the jQuery object relative to its sibling elements.

If `.index()` is called on a collection of elements and a DOM element or jQuery object is passed in, `.index()` returns an integer indicating the position of the passed element relative to the original collection.

If a selector string is passed as an argument, `.index()` returns an integer indicating the position of the original element relative to the elements matched by the selector. If the element is not found, `.index()` will return -1.

Detail

The complementary operation to `.get()`, which accepts an index and returns a DOM node, `.index()` can take a DOM node and returns an index. Suppose we have 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 retrieve 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');
alert('Index: ' + $('li').index(listItem));
We get back the zero-based position of the list item:
```

Index: 1

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

```
var listItem = $('#bar');
alert('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)');
alert('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 which also matches this selector is located.

```
var listItem = $('#bar');
alert('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:

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

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

Index: 1

Example
On click, returns the index (based zero) of that div in the page.

```
$("div").click(function () {
 // this is the dom element clicked
 var index = $("div").index(this);
 $("span").text("That was div index #" + index);
});
```

Example
Returns the index for the element with ID bar.

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

Example
Returns the index for the first item in the jQuery collection.

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

Example
Returns the index for the element with ID bar in relation to all <li> elements.

```
$('div').html('Index: ' + $('#bar').index('li') );
```

Example
Returns the index for the element with ID bar in relation to its siblings.

```
var barIndex = $('#bar').index();
$('div').html( 'Index: ' + barIndex );
```

Example
Returns -1, as there is no element with ID foobar.

```
var foobar = $("li").index( $('#foobar') );
$('div').html('Index: ' + foobar);
```

get([index])
Retrieve the DOM elements matched by the jQuery object.

Arguments
index - A zero-based integer indicating which element to retrieve.

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:

```
alert($('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:

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

Since 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:

```
alert($('li')[0]);
```

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

```
alert($('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">

Example

Selects all divs in the document and returns the DOM Elements as an Array, then uses the built-in reverse-method to reverse that array.

```
function disp(divs) {
   var a = [];
   for (var i = 0; i < divs.length; i++) {
    a.push(divs[i].innerHTML);
   }
   $("span").text(a.join(" "));
  }

  disp( $("div").get().reverse() );
```

Example

Gives the tag name of the element clicked on.

```
$("*", document.body).click(function (e) {
   e.stopPropagation();
   var domEl = $(this).get(0);
   $("span:first").text("Clicked on - " + domEl.tagName);
  });
```

size()

Return the number of elements in the jQuery object.

The `.size()` method is functionally equivalent to the `.length` property; however, **the** `.length` property is preferred because it does not have the overhead of a function call.

Given a simple unordered list on the page:

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

Both `.size()` and `.length` identify the number of items:

```
alert( "Size: " + $("li").size() );
alert( "Size: " + $("li").length );
```

This results in two alerts:

Size: 2

Size: 2

Example
Count the divs. Click to add more.

```
$(document.body)
.click(function() {
 $(this).append( $("<div>") );
 var n = $("div").size();
 $("span").text("There are " + n + " divs. Click to add more.");
})
// trigger the click to start
.click();
```

jQuery.noConflict([removeAll])
Relinquish jQuery's control of the $ variable.

Arguments
removeAll - A Boolean indicating whether to remove all jQuery variables from the global scope (including jQuery itself).

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 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.

Example
Maps the original object that was referenced by $ back to $.

```
jQuery.noConflict();
// Do something with jQuery
jQuery("div p").hide();
```

```
// Do something with another library's $()
$("content").style.display = 'none';
```

Example
Reverts the $ alias and then creates and executes a function to provide the $ as a jQuery alias inside the functions scope. Inside the function the original $ object is not available. This works well for most plugins that don't rely on any other library.

```
jQuery.noConflict();
(function($) {
 $(function() {
  // more code using $ as alias to jQuery
 });
})(jQuery);
// other code using $ as an alias to the other library
```

Example
You can chain the jQuery.noConflict() with the shorthand ready for a compact code.

```
jQuery.noConflict()(function(){
  // code using jQuery
});
// other code using $ as an alias to the other library
```

Example
Creates a different alias instead of jQuery to use in the rest of the script.

```
var j = jQuery.noConflict();
// Do something with jQuery
j("div p").hide();
// Do something with another library's $()
$("content").style.display = 'none';
```

Example
Completely move jQuery to a new namespace in another object.

```
var dom = {};
dom.query = jQuery.noConflict(true);
```

selected
Selects all elements that are selected.

The `:selected` selector works for `<option>` elements. It does not work for checkboxes or radio inputs; use `:checked` for them.

Example
Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.

```
$("select").change(function () {
     var str = "";
     $("select option:selected").each(function () {
        str += $(this).text() + " ";
       });
     $("div").text(str);
    })
    .trigger('change');
```

checked
Matches all elements that are checked.

The `:checked` selector works for checkboxes and radio buttons. For select elements, use the `:selected` selector.

Example
Finds all input elements that are checked.

```
function countChecked() {
 var n = $("input:checked").length;
 $("div").text(n + (n <= 1 ? " is" : " are") + " checked!");
}
countChecked();
$(":checkbox").click(countChecked);
```

Example

```
$("input").click(function() {
 $("#log").html( $(":checked").val() + " is checked!" );
});
```

disabled
Selects all elements that are disabled.

As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':disabled')` is equivalent to `$('*:disabled')`, so `$('input:disabled')` should be used instead.

Example
Finds all input elements that are disabled.

```
$("input:disabled").val("this is it");
```

enabled
Selects all elements that are enabled.

As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':enabled')` is equivalent to `$('*:enabled')`, so `$('input:enabled')` should be used instead.

Example
Finds all input elements that are enabled.

```
$("input:enabled").val("this is it");
```

file
Selects all elements of type file.

`:file` is equivalent to `[type="file"]`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':file')` is equivalent to `$('*:file')`, so `$('input:file')` should be used instead.

Example
Finds all file inputs.

```
var input = $("input:file").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

button
Selects all button elements and elements of type button.

An equivalent selector to `$(":button")` using valid CSS is `$("button, input[type='button']")`.

Example
Find all button inputs and mark them.

```
var input = $(":button").addClass("marked");
 $("div").text( "For this type jQuery found " + input.length + "." );
 $("form").submit(function () { return false; }); // so it won't submit
```

reset
Selects all elements of type reset.

`:reset` is equivalent to `[type="reset"]`

Example
Finds all reset inputs.

```
var input = $("input:reset").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

image
Selects all elements of type image.

`:image` is equivalent to `[type="image"]`

Example
Finds all image inputs.

```
var input = $("input:image").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

submit
Selects all elements of type submit.

The `:submit` selector typically applies to button or input elements. Note that some browsers treat `<button>` element as `type="default"` implicitly while others (such as Internet Explorer) do not.

Example
Finds all submit elements that are descendants of a td element.

```
var submitEl = $("td :submit")
   .parent('td')
   .css({background:"yellow", border:"3px red solid"})
  .end();

  $('#result').text('jQuery matched ' + submitEl.length + ' elements.');

  // so it won't submit
  $("form").submit(function () { return false; });
```

```
// Extra JS to make the HTML easier to edit (None of this is relevant to the ':submit' selector
$('#exampleTable').find('td').each(function(i, el) {
  var inputEl = $(el).children(),
    inputType = inputEl.attr('type') ? ' type="' + inputEl.attr('type') + '"' : '';
  $(el).before('<td>' + inputEl[0].nodeName + inputType + '</td>');
})
```

checkbox

Selects all elements of type checkbox.

`$(':checkbox')` is equivalent to `$('[type=checkbox]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':checkbox')` is equivalent to `$('*:checkbox')`, so `$('input:checkbox')` should be used instead.

Example

Finds all checkbox inputs.

```
var input = $("form input:checkbox").wrap('<span></span>').parent().css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

radio

Selects all elements of type radio.

`$(':radio')` is equivalent to `$('[type=radio]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':radio')` is equivalent to `$('*:radio')`, so `$('input:radio')` should be used instead.

To select a set of associated radio buttons, you might use: `$('input[name=gender]:radio')`

Example

Finds all radio inputs.

```
var input = $("form input:radio").wrap('<span></span>').parent().css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

password

Selects all elements of type password.

`$(':password')` is equivalent to `$('[type=password]')`. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':password')` is equivalent to `$('*:password')`, so `$('input:password')` should be used instead.

Example

Finds all password inputs.

```
var input = $("input:password").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

text
Selects all elements of type text.

`$(':text')` allows us to select all `<input type="text">` elements. As with other pseudo-class selectors (those that begin with a ":") it is recommended to precede it with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':text')` is equivalent to `$('*:text')`, so `$('input:text')` should be used instead.

Note: As of jQuery 1.5.2, `:text` selects `input` elements that have no specified `type` attribute (in which case `type="text"` is implied).

This difference in behavior between `$(':text')` and `$('[type=text]')`, can be seen below:

```
$('<input>').is('[type=text]'); // false
$('<input>').is(':text'); // true
```

Example
Finds all text inputs.

```
var input = $("form input:text").css({background:"yellow", border:"3px red solid"});
  $("div").text("For this type jQuery found " + input.length + ".")
      .css("color", "red");
  $("form").submit(function () { return false; }); // so it won't submit
```

input
Selects all input, textarea, select and button elements.

The `:input` selector basically selects all form controls.

Example
Finds all input elements.

```
var allInputs = $(":input");
  var formChildren = $("form > *");
  $("#messages").text("Found " + allInputs.length + " inputs and the form has " +
              formChildren.length + " children.");

  // so it won't submit
  $("form").submit(function () { return false; });
```

attributeContainsPrefix
Selects elements that have the specified attribute with a value either equal to a given string or starting with that string followed by a hyphen (-).

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector was introduced into the CSS specification to handle language attributes.

Example
Finds all links with an hreflang attribute that is english.

```
$('a[hreflang|="en"]').css('border','3px dotted green');
```

attributeContainsWord
Selects elements that have the specified attribute with a value containing a given word, delimited by spaces.

Arguments
attribute - An attribute name.

value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector matches the test string against each word in the attribute value, where a "word" is defined as a string delimited by whitespace. The selector matches if the test string is exactly equal to any of the words.

Example
Finds all inputs with a name attribute that contains the word 'man' and sets the value with some text.

```
$('input[name~="man"]').val('mr. man is in it!');
```

attributeMultiple
Matches elements that match all of the specified attribute filters.

Arguments
attributeFilter1 - An attribute filter.
attributeFilter2 - Another attribute filter, reducing the selection even more
attributeFilterN - As many more attribute filters as necessary

Example
Finds all inputs that have an id attribute and whose name attribute ends with man and sets the value.

```
$('input[id][name$="man"]').val('only this one');
```

attributeContains
Selects elements that have the specified attribute with a value containing the a given substring.

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This is the most generous of the jQuery attribute selectors that match against a value. It will select an element if the selector's string appears anywhere within the element's attribute value. Compare this selector with the Attribute Contains Word selector (e.g. [attr~="word"]), which is more appropriate in many cases.

Example
Finds all inputs with a name attribute that contains 'man' and sets the value with some text.

```
$('input[name*="man"]').val('has man in it!');
```

attributeEndsWith
Selects elements that have the specified attribute with a value ending exactly with a given string. The comparison is case sensitive.

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

Example
Finds all inputs with an attribute name that ends with 'letter' and puts text in them.

```
$('input[name$="letter"]').val('a letter');
```

attributeStartsWith
Selects elements that have the specified attribute with a value beginning exactly with a given string.

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector can be useful for identifying elements in pages produced by server-side frameworks that produce HTML with systematic element IDs.

However it will be slower than using a class selector so leverage classes, if you can, to group like elements.

Example
Finds all inputs with an attribute name that starts with 'news' and puts text in them.

```
$('input[name^="news"]').val('news here!');
```

attributeNotEqual
Select elements that either don't have the specified attribute, or do have the specified attribute but not with a certain value.

Arguments
attribute - An attribute name.
value - An attribute value. Can be either an unquoted single word or a quoted string.

This selector is equivalent to `:not([attr="value"])`.

Example
Finds all inputs that don't have the name 'newsletter' and appends text to the span next to it.

```
$('input[name!="newsletter"]').next().append('<b>; not newsletter</b>');
```

attributeEquals
Selects elements that have the specified attribute with a value exactly equal to a certain value.

Arguments
attribute - An attribute name.
value - An attribute value. **Can be either an unquoted single word or a quoted string.**

Example
Finds all inputs with a value of "Hot Fuzz" and changes the text of the next sibling span.

```
$('input[value="Hot Fuzz"]').next().text(" Hot Fuzz");
```

attributeHas
Selects elements that have the specified attribute, with any value.

Arguments
attribute - An attribute name.

Example
Bind a single click that adds the div id to its text.

```
$('div[id]').one('click', function(){
   var idString = $(this).text() + ' = ' + $(this).attr('id');
   $(this).text(idString);
  });
```

visible
Selects all elements that are visible.

Elements are considered visible if they consume space in the document. Visible elements have a width or height that is greater than zero.

Elements with `visibility: hidden` or `opacity: 0` are considered visible, since they still consume space in the layout. During animations that hide an element, the element is considered to be visible until the end of the animation. During animations to show an element, the element is considered to be visible at the start at the animation.

How `:visible` is calculated was changed in jQuery 1.3.2. The release notes outline the changes in more detail.

Example
Make all visible divs turn yellow on click.

```
$("div:visible").click(function () {
   $(this).css("background", "yellow");
  });
  $("button").click(function () {
   $("div:hidden").show("fast");
  });
```

hidden
Selects all elements that are hidden.

Elements can be considered hidden for several reasons:
- They have a CSS `display` value of `none`.
- They are form elements with `type="hidden"`.
- Their width and height are explicitly set to 0.
- An ancestor element is hidden, so the element is not shown on the page.

Elements with `visibility: hidden` or `opacity: 0` are considered to be visible, since they still consume space in the layout. During animations that hide an element, the element is considered to be visible until the end of the animation. During animations to show an element, the element is considered to be visible at the start of the animation.

How `:hidden` is determined was changed in jQuery 1.3.2. An element is assumed to be hidden if it or any of its parents consumes no space in the document. CSS visibility isn't taken into account (therefore `$(elem).css('visibility','hidden').is(':hidden') == false`). The release notes outline the changes in more detail.

Example
Shows all hidden divs and counts hidden inputs.

```
// in some browsers :hidden includes head, title, script, etc...
var hiddenEls = $("body").find(":hidden").not("script");

$("span:first").text("Found " + hiddenEls.length + " hidden elements total.");
$("div:hidden").show(3000);
$("span:last").text("Found " + $("input:hidden").length + " hidden inputs.");
```

parent
Select all elements that are the parent of another element, including text nodes.

This is the inverse of `:empty`.

One important thing to note regarding the use of `:parent` (and `:empty`) is that child elements include text nodes.

The W3C recommends that the <p> element have at least one child node, even if that child is merely text (see http://www.w3.org/TR/html401/struct/text.html#edef-P). Some other elements, on the other hand, are empty (i.e. have no children) by definition: <input>, <img>,
, and <hr>, for example.

Example
Finds all tds with children, including text.

```
$("td:parent").fadeTo(1500, 0.3);
```

empty
Select all elements that have no children (including text nodes).

This is the inverse of `:parent`.

One important thing to note with :empty (and :parent) is that child elements include text nodes.

The W3C recommends that the <p> element have at least one child node, even if that child is merely text (see http://www.w3.org/TR/html401/struct/text.html#edef-P). Some other elements, on the other hand, are empty (i.e. have no children) by definition: <input>, <img>,
, and <hr>, for example.

Example
Finds all elements that are empty - they don't have child elements or text.

```
$("td:empty").text("Was empty!").css('background', 'rgb(255,220,200)');
```

lt
Select all elements at an index less than `index` within the matched set.

Arguments
index - Zero-based index.

index-related selectors

The index-related selectors (including this "less than" selector) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:lt(1)')` selects the first element in the document with the class `myclass`, rather than selecting no elements. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Example
Finds TDs less than the one with the 4th index (TD#4).

```
$("td:lt(4)").css("color", "red");
```

gt
Select all elements at an index greater than `index` within the matched set.

Arguments
index - Zero-based index.

index-related selectors

The index-related selector expressions (including this "greater than" selector) filter the set of elements that have matched the expressions that precede them. They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices 0 through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:gt(1)')` selects elements after the second element in the document with the class `myclass`, rather than after the first. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Example
Finds TD #5 and higher. Reminder: the indexing starts at 0.

```
$("td:gt(4)").css("text-decoration", "line-through");
```

eq
Select the element at index n within the matched set.

Arguments
index - Zero-based index of the element to match.

The index-related selectors (`:eq()`, `:lt()`, `:gt()`, `:even`, `:odd`) filter the set of elements that have matched the expressions that precede them.

They narrow the set down based on the order of the elements within this matched set. For example, if elements are first selected with a class selector (`.myclass`) and four elements are returned, these elements are given indices `0` through 3 for the purposes of these selectors.

Note that since JavaScript arrays use *0-based indexing*, these selectors reflect that fact. This is why `$('.myclass:eq(1)')` selects the second element in the document with the class myclass, rather than the first. In contrast, `:nth-child(n)` uses *1-based indexing* to conform to the CSS specification.

Unlike the .eq(index) method, the `:eq(index)` selector does *not* accept a negative value for `index`. For example, while `$('li').eq(-1)` selects the last `li` element, `$('li:eq(-1)')` selects nothing.

Example
Finds the third td.

```
$("td:eq(2)").css("color", "red");
```

Example
Apply three different styles to list items to demonstrate that `:eq()` is designed to select a single element while `:nth-child()` or `:eq()` within a looping construct such as `.each()` can select multiple elements.

```
// applies yellow background color to a single <li>
$("ul.nav li:eq(1)").css( "backgroundColor", "#ff0" );

// applies italics to text of the second <li> within each <ul class="nav">
$("ul.nav").each(function(index) {
 $(this).find("li:eq(1)").css( "fontStyle", "italic" );
});

// applies red text color to descendants of <ul class="nav">
// for each <li> that is the second child of its parent
$("ul.nav li:nth-child(2)").css( "color", "red" );
```

odd
Selects odd elements, zero-indexed. See also even.

In particular, note that the *0-based indexing* means that, counter-intuitively, `:odd` selects the second element, fourth element, and so on within the matched set.

Example
Finds odd table rows, matching the second, fourth and so on (index 1, 3, 5 etc.).

```
$("tr:odd").css("background-color", "#bbbbff");
```

even
Selects even elements, zero-indexed. See also odd.

In particular, note that the *0-based indexing* means that, counter-intuitively, `:even` selects the first element, third element, and so on within the matched set.

Example
Finds even table rows, matching the first, third and so on (index 0, 2, 4 etc.).

```
$("tr:even").css("background-color", "#bbbbff");
```

not
Selects all elements that do not match the given selector.

Arguments
selector - A selector with which to filter by.

All selectors are accepted inside `:not()`, for example: `:not(div a)` and `:not(div,a)`.

Additional Notes

The `.not()` method will end up providing you with more readable selections than pushing complex selectors or variables into a `:not()` selector filter. In most cases, it is a better choice.

Example
Finds all inputs that are not checked and highlights the next sibling span. Notice there is no change when clicking the checkboxes since no click events have been linked.

```
$("input:not(:checked) + span").css("background-color", "yellow");
 $("input").attr("disabled", "disabled");
```

last
Selects the last matched element.

Note that `:last` selects a single element by filtering the current jQuery collection and matching the last element within it.

Example
Finds the last table row.

```
$("tr:last").css({backgroundColor: 'yellow', fontWeight: 'bolder'});
```

first
Selects the first matched element.

The `:first` pseudo-class is equivalent to `:eq(0)`. It could also be written as `:lt(1)`. While this matches only a single element, :first-child can match more than one: One for each parent.

Example
Finds the first table row.

```
$("tr:first").css("font-style", "italic");
```

next siblings
Selects all sibling elements that follow after the "prev" element, have the same parent, and match the filtering "siblings" selector.

Arguments
prev - Any valid selector.
siblings - A selector to filter elements that are the following siblings of the first selector.

The notable difference between (`prev + next`) and (`prev ~ siblings`) is their respective reach. While the former reaches only to the immediately following sibling element, the latter extends that reach to all following sibling elements.

Example
Finds all divs that are siblings after the element with #prev as its id. Notice the span isn't selected since it is not a div and the "niece" isn't selected since it is a child of a sibling, not an actual sibling.

```
$("#prev ~ div").css("border", "3px groove blue");
```

next adjacent
Selects all next elements matching "next" that are immediately preceded by a sibling "prev".

Arguments
prev - Any valid selector.

next - A selector to match the element that is next to the first selector.

One important point to consider with both the next adjacent sibling selector (`prev + next`) and the general sibling selector (`prev ~ siblings`) is that the elements on either side of the combinator must share the same parent.

Example
Finds all inputs that are next to a label.

```
$("label + input").css("color", "blue").val("Labeled!")
```

child
Selects all direct child elements specified by "child" of elements specified by "parent".

Arguments
parent - Any valid selector.
child - A selector to filter the child elements.

As a CSS selector, the child combinator is supported by all modern web browsers including Safari, Firefox, Opera, Chrome, and Internet Explorer 7 and above, but notably not by Internet Explorer versions 6 and below. However, in jQuery, this selector (along with all others) works across all supported browsers, including IE6.

The child combinator (E **>** F) can be thought of as a more specific form of the descendant combinator (E F) in that it selects only first-level descendants.

Note: The `$("> elem", context)` selector will be deprecated in a future release. Its usage is thus discouraged in lieu of using alternative selectors.

Example
Places a border around all list items that are children of <ul class="topnav"> .

```
$("ul.topnav > li").css("border", "3px double red");
```

descendant
Selects all elements that are descendants of a given ancestor.

Arguments
ancestor - Any valid selector.
descendant - A selector to filter the descendant elements.

A descendant of an element could be a child, grandchild, great-grandchild, and so on, of that element.

Example
Finds all input descendants of forms.

```
$("form input").css("border", "2px dotted blue");
```

multiple
Selects the combined results of all the specified selectors.

Arguments
selector1 - Any valid selector.
selector2 - Another valid selector.
selectorN - As many more valid selectors as you like.

You can specify any number of selectors to combine into a single result. This multiple expression combinator is an efficient way to select disparate elements. The order of the DOM elements in the returned jQuery object may not be identical, as they will be in document order. An alternative to this combinator is the .add() method.

Example
Finds the elements that match any of these three selectors.

```
$("div,span,p.myClass").css("border","3px solid red");
```

Example
Show the order in the jQuery object.

```
var list = $("div,p,span").map(function () {
   return this.tagName;
  }).get().join(", ");
  $("b").append(document.createTextNode(list));
```

all
Selects all elements.

Caution: The all, or universal, selector is extremely slow, except when used by itself.

Example
Finds every element (including head, body, etc) in the document.

```
var elementCount = $("*").css("border","3px solid red").length;
$("body").prepend("<h3>" + elementCount + " elements found</h3>");
```

Example
A common way to select all elements is to find within document.body so elements like head, script, etc are left out.

```
var elementCount = $("#test").find("*").css("border","3px solid red").length;
$("body").prepend("<h3>" + elementCount + " elements found</h3>");
```

class
Selects all elements with the given class.

Arguments
class - A class to search for. An element can have multiple classes; only one of them must match.

For class selectors, jQuery uses JavaScript's native `getElementsByClassName()` function if the browser supports it.

Example
Finds the element with the class "myClass".

```
$(".myClass").css("border","3px solid red");
```

Example
Finds the element with both "myclass" and "otherclass" classes.

```
$(".myclass.otherclass").css("border","13px solid red");
```

element
Selects all elements with the given tag name.

Arguments
element - An element to search for. Refers to the tagName of DOM nodes.

JavaScript's `getElementsByTagName()` function is called to return the appropriate elements when this expression is used.

Example
Finds every DIV element.

```
$("div").css("border","9px solid red");
```

id
Selects a single element with the given id attribute.

Arguments
id - An ID to search for, specified via the id attribute of an element.

For id selectors, jQuery uses the JavaScript function `document.getElementById()`, which is extremely efficient. When another selector is attached to the id selector, such as `h2#pageTitle`, jQuery performs an additional check before identifying the element as a match.

As always, remember that as a developer, your time is typically the most valuable resource. Do not focus on optimization of selector speed unless it is clear that performance needs to be improved.

Each `id` value must be used only once within a document. If more than one element has been assigned the same ID, queries that use that ID will only select the first matched element in the DOM. This behavior should not be relied on, however; a document with more than one element using the same ID is invalid.

If the id contains characters like periods or colons you have to escape those characters with backslashes.

Example
Finds the element with the id "myDiv".

```
$("#myDiv").css("border","3px solid red");
```

Example
Finds the element with the id "myID.entry[1]". See how certain characters must be escaped with backslashes.

```
$("#myID\.entry\[1\]").css("border","3px solid red");
```

scroll(handler(eventObject))
Bind an event handler to the "scroll" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('scroll', handler)` in the first and second variations, and `.trigger('scroll')` in the third.

The `scroll` event is sent to an element when the user scrolls to a different place in the element. It applies to `window` objects, but also to scrollable frames and elements with the `overflow` CSS property set to `scroll` (or `auto` when the element's explicit height or width is less than the height or width of its contents).

For example, consider the HTML:

```
<div id="target" style="overflow: scroll; width: 200px; height: 100px;">
 Lorem ipsum dolor sit amet, consectetur adipisicing elit,
 sed do eiusmod tempor incididunt ut labore et dolore magna
 aliqua. Ut enim ad minim veniam, quis nostrud exercitation
 ullamco laboris nisi ut aliquip ex ea commodo consequat.
 Duis aute irure dolor in reprehenderit in voluptate velit
 esse cillum dolore eu fugiat nulla pariatur. Excepteur
 sint occaecat cupidatat non proident, sunt in culpa qui
 officia deserunt mollit anim id est laborum.
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The style definition is present to make the target element small enough to be scrollable:

The `scroll` event handler can be bound to this element:

```
$('#target').scroll(function() {
 $('#log').append('<div>Handler for .scroll() called.</div>');
});
```

Now when the user scrolls the text up or down, one or more messages are appended to `<div id="log"></div>`:

Handler for .scroll() called.

To trigger the event manually, apply `.scroll()` without an argument:

```
$('#other').click(function() {
 $('#target').scroll();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

A `scroll` event is sent whenever the element's scroll position changes, regardless of the cause. A mouse click or drag on the scroll bar, dragging inside the element, pressing the arrow keys, or using the mouse's scroll wheel could cause this event.

Example
To do something when your page is scrolled:

```
$("p").clone().appendTo(document.body);
  $("p").clone().appendTo(document.body);
  $("p").clone().appendTo(document.body);
  $(window).scroll(function () {
   $("span").css("display", "inline").fadeOut("slow");
  });
```

resize(handler(eventObject))
Bind an event handler to the "resize" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('resize', handler)` in the first and second variations, and `.trigger('resize')` in the third.

The `resize` event is sent to the `window` element when the size of the browser window changes:

```
$(window).resize(function() {
 $('#log').append('<div>Handler for .resize() called.</div>');
});
```

Now whenever the browser window's size is changed, the message is appended to one or more times, depending on the browser.

Code in a `resize` handler should never rely on the number of times the handler is called. Depending on implementation, `resize` events can be sent continuously as the resizing is in progress (the typical behavior in Internet Explorer and WebKit-based browsers such as Safari and Chrome), or only once at the end of the resize operation (the typical behavior in some other browsers such as Opera).

Example
To see the window width while (or after) it is resized, try:

```
$(window).resize(function() {
 $('body').prepend('<div>' + $(window).width() + '</div>');
});
```

keyup(handler(eventObject))
Bind an event handler to the "keyup" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('keyup', handler)` in the first two variations, and `.trigger('keyup')` in the third.

The `keyup` event is sent to an element when the user releases a key on the keyboard. It can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$('#target').keyup(function() {
 alert('Handler for .keyup() called.');
});
```

Now when the insertion point is inside the field and a key is pressed and released, the alert is displayed:

Handler for .keyup() called.

To trigger the event manually, apply `.keyup()` without arguments:

```
$('#other').click(function() {
 $('#target').keyup();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which key was pressed, examine the event object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the key code. This code corresponds to a key on the keyboard, including codes for special keys such as arrows. For catching actual text entry, `.keypress()` may be a better choice.

Example

Show the event object for the keyup handler (using a simple $.print plugin) when a key is released in the input.

```
var xTriggered = 0;
$('#target').keyup(function(event) {
  xTriggered++;
  var msg = 'Handler for .keyup() called ' + xTriggered + ' time(s).';
 $.print(msg, 'html');
 $.print(event);
}).keydown(function(event) {
 if (event.which == 13) {
  event.preventDefault();
 }
});

$('#other').click(function() {
 $('#target').keyup();
});
```

keypress(handler(eventObject))

Bind an event handler to the "keypress" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

Note: as the `keypress` event isn't covered by any official specification, the actual behavior encountered when using it may differ across browsers, browser versions, and platforms.

This method is a shortcut for `.bind("keypress", handler)` in the first two variations, and `.trigger("keypress")` in the third.

The `keypress` event is sent to an element when the browser registers keyboard input. This is similar to the `keydown` event, except in the case of key repeats. If the user presses and holds a key, a `keydown` event is triggered once, but separate `keypress` events are triggered for each inserted character. In addition, modifier keys (such as Shift) trigger `keydown` events but not `keypress` events.

A `keypress` event handler can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <fieldset>
  <input id="target" type="text" value="Hello there" />
 </fieldset>
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$("#target").keypress(function() {
 alert("Handler for .keypress() called.");
});
```

Now when the insertion point is inside the field, pressing a key displays the alert:

Handler for .keypress() called.

The message repeats if the key is held down. To trigger the event manually, apply `.keypress()` without an argument::

```
$('#other').click(function() {
 $("#target").keypress();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which character was entered, examine the `event` object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the character code.

Note that `keydown` and `keyup` provide a code indicating which key is pressed, while `keypress` indicates which character was entered. For example, a lowercase "a" will be reported as 65 by `keydown` and `keyup`, but as 97 by `keypress`. An uppercase "A" is reported as 65 by all events. Because of this distinction, when catching special keystrokes such as arrow keys, `.keydown()` or `.keyup()` is a better choice.

Example

Show the event object when a key is pressed in the input. Note: This demo relies on a simple $.print() plugin (http://api.jquery.com/scripts/events.js) for the event object's output.

```
var xTriggered = 0;
$("#target").keypress(function(event) {
```

```
 if ( event.which == 13 ) {
   event.preventDefault();
  }
  xTriggered++;
  var msg = "Handler for .keypress() called " + xTriggered + " time(s).";
 $.print( msg, "html" );
 $.print( event );
});

$("#other").click(function() {
 $("#target").keypress();
});
```

submit(handler(eventObject))

Bind an event handler to the "submit" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('submit', handler)` in the first variation, and `.trigger('submit')` in the third.

The `submit` event is sent to an element when the user is attempting to submit a form. It can only be attached to `<form>` elements. Forms can be submitted either by clicking an explicit `<input type="submit">`, `<input type="image">`, or `<button type="submit">`, or by pressing Enter when certain form elements have focus.

Depending on the browser, the Enter key may only cause a form submission if the form has exactly one text field, or only when there is a submit button present. The interface should not rely on a particular behavior for this key unless the issue is forced by observing the keypress event for presses of the Enter key.

For example, consider the HTML:

```
<form id="target" action="destination.html">
 <input type="text" value="Hello there" />
 <input type="submit" value="Go" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the form:

```
$('#target').submit(function() {
 alert('Handler for .submit() called.');
 return false;
});
```

Now when the form is submitted, the message is alerted. This happens prior to the actual submission, so we can cancel the submit action by calling `.preventDefault()` on the event object or by returning `false` from our handler. We can trigger the event manually when another element is clicked:

```
$('#other').click(function() {
 $('#target').submit();
});
```

After this code executes, clicks on Trigger the handler will also display the message. In addition, the default `submit` action on the form will be fired, so the form will be submitted.

The JavaScript `submit` event does not bubble in Internet Explorer. However, scripts that rely on event delegation with the `submit` event will work consistently across browsers as of jQuery 1.4, which has normalized the event's behavior.

Example

If you'd like to prevent forms from being submitted unless a flag variable is set, try:

```
$("form").submit(function() {
   if ($("input:first").val() == "correct") {
    $("span").text("Validated...").show();
    return true;
   }
   $("span").text("Not valid!").show().fadeOut(1000);
   return false;
  });
```

Example
If you'd like to prevent forms from being submitted unless a flag variable is set, try:

```
$("form").submit( function () {
 return this.some_flag_variable;
} );
```

Example
To trigger the submit event on the first form on the page, try:

```
$("form:first").submit();
```

select(handler(eventObject))
Bind an event handler to the "select" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('select', handler)` in the first two variations, and `.trigger('select')` in the third.

The `select` event is sent to an element when the user makes a text selection inside it. This event is limited to `<input type="text">` fields and `<textarea>` boxes.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the text input:

```
$('#target').select(function() {
 alert('Handler for .select() called.');
});
```

Now when any portion of the text is selected, the alert is displayed. Merely setting the location of the insertion point will not trigger the event. To trigger the event manually, apply `.select()` without an argument:

```
$('#other').click(function() {
 $('#target').select();
});
```

After this code executes, clicks on the Trigger button will also alert the message:

Handler for .select() called.

In addition, the default `select` action on the field will be fired, so the entire text field will be selected.

The method for retrieving the current selected text differs from one browser to another. A number of jQuery plug-ins offer cross-platform solutions.

Example
To do something when text in input boxes is selected:

```
$(":input").select( function () {
   $("div").text("Something was selected").show().fadeOut(1000);
  });
```

Example
To trigger the select event on all input elements, try:

```
$("input").select();
```

change(handler(eventObject))
Bind an event handler to the "change" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('change', handler)` in the first two variations, and `.trigger('change')` in the third.

The `change` event is sent to an element when its value changes. This event is limited to `<input>` elements, `<textarea>` boxes and `<select>` elements. For select boxes, checkboxes, and radio buttons, the event is fired immediately when the user makes a selection with the mouse, but for the other element types the event is deferred until the element loses focus.

For example, consider the HTML:

```
<form>
 <input class="target" type="text" value="Field 1" />
 <select class="target">
  <option value="option1" selected="selected">Option 1</option>
  <option value="option2">Option 2</option>
 </select>
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the text input and the select box:

```
$('.target').change(function() {
 alert('Handler for .change() called.');
});
```

Now when the second option is selected from the dropdown, the alert is displayed. It is also displayed if you change the text in the field and then click away. If the field loses focus without the contents having changed, though, the event is not triggered. To trigger the event manually, apply `.change()` without arguments:

```
$('#other').click(function() {
 $('.target').change();
});
```

After this code executes, clicks on Trigger the handler will also alert the message. The message will display twice, because the handler has been bound to the `change` event on both of the form elements.

As of jQuery 1.4, the `change` event bubbles in Internet Explorer, behaving consistently with the event in other modern browsers.

Example
Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.

```
$("select").change(function () {
     var str = "";
```

```
    $("select option:selected").each(function () {
       str += $(this).text() + " ";
      });
    $("div").text(str);
   })
   .change();
```

Example
To add a validity test to all text input elements:

```
$("input[type='text']").change( function() {
 // check input ($(this).val()) for validity here
});
```

blur(handler(eventObject))
Bind an event handler to the "blur" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('blur', handler)` in the first two variations, and `.trigger('blur')` in the third.

The `blur` event is sent to an element when it loses focus. Originally, this event was only applicable to form elements, such as `<input>`. In recent browsers, the domain of the event has been extended to include all element types. An element can lose focus via keyboard commands, such as the Tab key, or by mouse clicks elsewhere on the page.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Field 1" />
 <input type="text" value="Field 2" />
</form>
<div id="other">
 Trigger the handler
</div>
The event handler can be bound to the first input field:
$('#target').blur(function() {
 alert('Handler for .blur() called.');
});
```

Now if the first field has the focus, clicking elsewhere or tabbing away from it displays the alert:

Handler for .blur() called.

To trigger the event programmatically, apply `.blur()` without an argument:

```
$('#other').click(function() {
 $('#target').blur();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `blur` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `blur` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `blur` to the `focusout` event in its event delegation methods, .live() and .delegate().

Example
To trigger the blur event on all paragraphs:

```
$("p").blur();
```

focus(handler(eventObject))

Bind an event handler to the "focus" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

• This method is a shortcut for `.bind('focus', handler)` in the first and second variations, and `.trigger('focus')` in the third.

• The `focus` event is sent to an element when it gains focus. This event is implicitly applicable to a limited set of elements, such as form elements (`<input>`, `<select>`, etc.) and links (`<a href>`). In recent browser versions, the event can be extended to include all element types by explicitly setting the element's `tabindex` property. An element can gain focus via keyboard commands, such as the Tab key, or by mouse clicks on the element.

• Elements with focus are usually highlighted in some way by the browser, for example with a dotted line surrounding the element. The focus is used to determine which element is the first to receive keyboard-related events.

Attempting to set focus to a hidden element causes an error in Internet Explorer. Take care to only use `.focus()` on elements that are visible. To run an element's focus event handlers without setting focus to the element, use `.triggerHandler("focus")` instead of `.focus()`.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Field 1" />
 <input type="text" value="Field 2" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the first input field:

```
$('#target').focus(function() {
 alert('Handler for .focus() called.');
});
```

Now clicking on the first field, or tabbing to it from another field, displays the alert:

Handler for .focus() called.

We can trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#target').focus();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `focus` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `focus` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `focus` to the `focusin` event in its event delegation methods, .live() and .delegate().

Example

Fire focus.

```
$("input").focus(function () {
     $(this).next("span").css('display','inline').fadeOut(1000);
  });
```

Example

To stop people from writing in text input boxes, try:

```
$("input[type=text]").focus(function(){
 $(this).blur();
});
```

Example
To focus on a login input box with id 'login' on page startup, try:

```
$(document).ready(function(){
 $("#login").focus();
});
```

mousemove(handler(eventObject))
Bind an event handler to the "mousemove" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mousemove', handler)` in the first two variations, and `.trigger('mousemove')` in the third.

The `mousemove` event is sent to an element when the mouse pointer moves inside the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Move here
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to the target:

```
$("#target").mousemove(function(event) {
 var msg = "Handler for .mousemove() called at ";
 msg += event.pageX + ", " + event.pageY;
 $("#log").append("<div>" + msg + "</div>");
});
```

Now when the mouse pointer moves within the target button, the messages are appended to :

Handler for .mousemove() called at (399, 48)Handler for .mousemove() called at (398, 46)Handler for .mousemove() called at (397, 44)Handler for .mousemove() called at (396, 42)

To trigger the event manually, apply `.mousemove()` without an argument:

```
$("#other").click(function() {
 $("#target").mousemove();
});
```

After this code executes, clicks on the Trigger button will also append the message:

Handler for .mousemove() called at (undefined, undefined)

When tracking mouse movement, you usually need to know the actual position of the mouse pointer. The event object that is passed to the handler contains some information about the mouse coordinates. Properties such as `.clientX`, `.offsetX`, and `.pageX` are available, but support for them differs between browsers. Fortunately, jQuery normalizes the `.pageX` and `.pageY` properties so that they can be used in all browsers. These properties provide the X and Y coordinates of the mouse pointer relative to the top-left corner of the document, as illustrated in the example output above.

Keep in mind that the `mousemove` event is triggered whenever the mouse pointer moves, even for a pixel. This means that hundreds of events can be generated over a very small amount of time. If the handler has to do any significant processing, or if multiple handlers for the event exist, this can be a serious performance drain on the browser. It is important, therefore, to optimize `mousemove` handlers as much as possible, and to unbind them as soon as they are no longer needed.

A common pattern is to bind the `mousemove` handler from within a `mousedown` hander, and to unbind it from a corresponding `mouseup` handler. If

implementing this sequence of events, remember that the `mouseup` event might be sent to a different HTML element than the `mousemove` event was. To account for this, the `mouseup` handler should typically be bound to an element high up in the DOM tree, such as `<body>`.

Example
Show the mouse coordinates when the mouse is moved over the yellow div. Coordinates are relative to the window, which in this case is the iframe.

```
$("div").mousemove(function(e){
   var pageCoords = "( " + e.pageX + ", " + e.pageY + " )";
   var clientCoords = "( " + e.clientX + ", " + e.clientY + " )";
   $("span:first").text("( e.pageX, e.pageY ) : " + pageCoords);
   $("span:last").text("( e.clientX, e.clientY ) : " + clientCoords);
  });
```

hover(handlerIn(eventObject), handlerOut(eventObject))
Bind two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

Arguments
handlerIn(eventObject) - A function to execute when the mouse pointer enters the element.
handlerOut(eventObject) - A function to execute when the mouse pointer leaves the element.

The `.hover()` method binds handlers for both `mouseenter` and `mouseleave` events. You can use it to simply apply behavior to an element during the time the mouse is within the element.

Calling `$(selector).hover(handlerIn, handlerOut)` is shorthand for:

```
$(selector).mouseenter(handlerIn).mouseleave(handlerOut);
```

See the discussions for `.mouseenter()` and `.mouseleave()` for more details.

Example
To add a special style to list items that are being hovered over, try:

```
$("li").hover(
 function () {
  $(this).append($("<span> ***</span>"));
 },
 function () {
  $(this).find("span:last").remove();
 }
);



//li with fade class
$("li.fade").hover(function(){$(this).fadeOut(100);$(this).fadeIn(500);});
```

Example
To add a special style to table cells that are being hovered over, try:

```
$("td").hover(
 function () {
  $(this).addClass("hover");
 },
 function () {
  $(this).removeClass("hover");
 }
);
```

Example
To unbind the above example use:

```
$("td").unbind('mouseenter mouseleave');
```

hover(handlerInOut(eventObject))

Bind a single handler to the matched elements, to be executed when the mouse pointer enters or leaves the elements.

Arguments

handlerInOut(eventObject) - A function to execute when the mouse pointer enters or leaves the element.

The `.hover()` method, when passed a single function, will execute that handler for both `mouseenter` and `mouseleave` events. This allows the user to use jQuery's various toggle methods within the handler or to respond differently within the handler depending on the `event.type`.

Calling `$(selector).hover(handlerInOut)` is shorthand for:

```
$(selector).bind("mouseenter mouseleave", handlerInOut);
```

See the discussions for `.mouseenter()` and `.mouseleave()` for more details.

Example

Slide the next sibling LI up or down on hover, and toggle a class.

```
$("li")
.filter(":odd")
.hide()
 .end()
.filter(":even")
.hover(
 function () {
  $(this).toggleClass("active")
   .next().stop(true, true).slideToggle();
 }
);
```

mouseleave(handler(eventObject))

Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseleave', handler)` in the first two variations, and `.trigger('mouseleave')` in the third.

The `mouseleave` JavaScript event is proprietary to Internet Explorer. Because of the event's general utility, jQuery simulates this event so that it can be used regardless of browser. This event is sent to an element when the mouse pointer leaves the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseleave(function() {
 $('#log').append('<div>Handler for .mouseleave() called.</div>');
});
```

Now when the mouse pointer moves out of the Outer , the message is appended to . You can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseleave();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

The mouseleave event differs from mouseout in the way it handles event bubbling. If mouseout were used in this example, then when the mouse pointer moved out of the Inner element, the handler would be triggered. This is usually undesirable behavior. The mouseleave event, on the other hand, only triggers its handler when the mouse leaves the element it is bound to, not a descendant. So in this example, the handler is triggered when the mouse leaves the Outer element, but not the Inner element.

Example

Show number of times mouseout and mouseleave events are triggered.mouseout fires when the pointer moves out of child element as well, while mouseleave fires only when the pointer moves out of the bound element.

```
var i = 0;
  $("div.overout").mouseover(function(){
   $("p:first",this).text("mouse over");
  }).mouseout(function(){
   $("p:first",this).text("mouse out");
   $("p:last",this).text(++i);
  });

  var n = 0;
  $("div.enterleave").mouseenter(function(){
   $("p:first",this).text("mouse enter");
  }).mouseleave(function(){
   $("p:first",this).text("mouse leave");
   $("p:last",this).text(++n);
  });
```

mouseenter(handler(eventObject))

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for .bind('mouseenter', handler) in the first two variations, and .trigger('mouseenter') in the third.

The mouseenter JavaScript event is proprietary to Internet Explorer. Because of the event's general utility, jQuery simulates this event so that it can be used regardless of browser. This event is sent to an element when the mouse pointer enters the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseenter(function() {
 $('#log').append('<div>Handler for .mouseenter() called.</div>');
});
```

Now when the mouse pointer moves over the Outer , the message is appended to . You can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseenter();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

The mouseenter event differs from mouseover in the way it handles event bubbling. If mouseover were used in this example, then when the mouse pointer moved over the Inner element, the handler would be triggered. This is usually undesirable behavior. The mouseenter event, on the other hand, only triggers its handler when the mouse enters the element it is bound to, not a descendant. So in this example, the handler is triggered when the mouse enters the Outer element, but not the Inner element.

Example

Show texts when mouseenter and mouseout event triggering.mouseover fires when the pointer moves into the child element as well, while mouseenter fires only when the pointer moves into the bound element.

```
var i = 0;
  $("div.overout").mouseover(function(){
   $("p:first",this).text("mouse over");
   $("p:last",this).text(++i);
  }).mouseout(function(){
   $("p:first",this).text("mouse out");
  });

  var n = 0;
  $("div.enterleave").mouseenter(function(){
   $("p:first",this).text("mouse enter");
   $("p:last",this).text(++n);
  }).mouseleave(function(){
   $("p:first",this).text("mouse leave");
  });
```

mouseout(handler(eventObject))

Bind an event handler to the "mouseout" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for .bind('mouseout', handler) in the first two variation, and .trigger('mouseout') in the third.

The mouseout event is sent to an element when the mouse pointer leaves the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseout(function() {
 $('#log').append('Handler for .mouseout() called.');
});
```

Now when the mouse pointer moves out of the Outer , the message is appended to . To trigger the event manually, apply .mouseout() without an argument::

```
$('#other').click(function() {
 $('#outer').mouseout();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

This event type can cause many headaches due to event bubbling. For instance, when the mouse pointer moves out of the Inner element in this example, a mouseout event will be sent to that, then trickle up to Outer. This can trigger the bound mouseout handler at inopportune times. See the discussion for .mouseleave() for a useful alternative.

Example

Show the number of times mouseout and mouseleave events are triggered.mouseout fires when the pointer moves out of the child element as well, while mouseleave fires only when the pointer moves out of the bound element.

```
var i = 0;
  $("div.overout").mouseout(function(){
   $("p:first",this).text("mouse out");
   $("p:last",this).text(++i);
  }).mouseover(function(){
   $("p:first",this).text("mouse over");
  });

  var n = 0;
  $("div.enterleave").bind("mouseenter",function(){
   $("p:first",this).text("mouse enter");
  }).bind("mouseleave",function(){
   $("p:first",this).text("mouse leave");
   $("p:last",this).text(++n);
  });
```

mouseover(handler(eventObject))

Bind an event handler to the "mouseover" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for .bind('mouseover', handler) in the first two variations, and .trigger('mouseover') in the third.

The mouseover event is sent to an element when the mouse pointer enters the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseover(function() {
 $('#log').append('<div>Handler for .mouseover() called.</div>');
});
```

Now when the mouse pointer moves over the Outer , the message is appended to . We can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseover();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

This event type can cause many headaches due to event bubbling. For instance, when the mouse pointer moves over the Inner element in this example, a `mouseover` event will be sent to that, then trickle up to Outer. This can trigger our bound `mouseover` handler at inopportune times. See the discussion for `.mouseenter()` for a useful alternative.

Example

Show the number of times mouseover and mouseenter events are triggered.`mouseover` fires when the pointer moves into the child element as well, while `mouseenter` fires only when the pointer moves into the bound element.

```
var i = 0;
 $("div.overout").mouseover(function() {
  i += 1;
  $(this).find("span").text( "mouse over x " + i );
 }).mouseout(function(){
  $(this).find("span").text("mouse out ");
 });

 var n = 0;
 $("div.enterleave").mouseenter(function() {
  n += 1;
  $(this).find("span").text( "mouse enter x " + n );
 }).mouseleave(function() {
  $(this).find("span").text("mouse leave");
 });
```

dblclick(handler(eventObject))

Bind an event handler to the "dblclick" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('dblclick', handler)` in the first two variations, and `.trigger('dblclick')` in the third.The `dblclick` event is sent to an element when the element is double-clicked. Any HTML element can receive this event.For example, consider the HTML:

```
<div id="target">
 Double-click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any :

```
$('#target').dblclick(function() {
 alert('Handler for .dblclick() called.');
});
```

Now double-clicking on this element displays the alert:

Handler for .dblclick() called.

To trigger the event manually, apply `.dblclick()` without an argument:

```
$('#other').click(function() {
 $('#target').dblclick();
});
```

After this code executes, (single) clicks on Trigger the handler will also alert the message.

The `dblclick` event is only triggered after this exact series of events:

- The mouse button is depressed while the pointer is inside the element.
- The mouse button is released while the pointer is inside the element.
- The mouse button is depressed again while the pointer is inside the element, within a time window that is system-dependent.
- The mouse button is released while the pointer is inside the element.

It is inadvisable to bind handlers to both the `click` and `dblclick` events for the same element. The sequence of events triggered varies from browser to browser, with some receiving two `click` events before the `dblclick` and others only one. Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

Example

To bind a "Hello World!" alert box the dblclick event on every paragraph on the page:

```
$("p").dblclick( function () { alert("Hello World!"); });
```

Example

Double click to toggle background color.

```
var divdbl = $("div:first");
  divdbl.dblclick(function () {
   divdbl.toggleClass('dbl');
  });
```

click(handler(eventObject))

Bind an event handler to the "click" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

In the first two variations, this method is a shortcut for `.bind("click", handler)`, as well as for `.on("click", handler)` as of jQuery 1.7. In the third variation, when `.click()` is called without arguments, it is a shortcut for `.trigger("click")`.

The `click` event is sent to an element when the mouse pointer is over the element, and the mouse button is pressed and released. Any HTML element can receive this event.

```
For example, consider the HTML:
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$("#target").click(function() {
 alert("Handler for .click() called.");
});
```

Now if we click on this element, the alert is displayed:

Handler for .click() called.

We can also trigger the event when a different element is clicked:

```
$("#other").click(function() {
 $("#target").click();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `click` event is only triggered after this exact series of events:
- The mouse button is depressed while the pointer is inside the element.
- The mouse button is released while the pointer is inside the element.

This is usually the desired sequence before taking an action. If this is not required, the `mousedown` or `mouseup` event may be more suitable.

Example
Hide paragraphs on a page when they are clicked:

```
$("p").click(function () {
   $(this).slideUp();
  });
```

Example
Trigger the click event on all of the paragraphs on the page:

```
$("p").click();
```

mouseup(handler(eventObject))
Bind an event handler to the "mouseup" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseup', handler)` in the first variation, and `.trigger('mouseup')` in the second.

The `mouseup` event is sent to an element when the mouse pointer is over the element, and the mouse button is released. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$('#target').mouseup(function() {
 alert('Handler for .mouseup() called.');
});
```

Now if we click on this element, the alert is displayed:

Handler for .mouseup() called.

We can also trigger the event when a different element is clicked:

```
$('#other').click(function() {
 $('#target').mouseup();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If the user clicks outside an element, drags onto it, and releases the button, this is still counted as a `mouseup` event. This sequence of actions is not treated as a button press in most user interfaces, so it is usually better to use the `click` event unless we know that the `mouseup` event is preferable for a particular situation.

Example

Show texts when mouseup and mousedown event triggering.

```
$("p").mouseup(function(){
   $(this).append('<span style="color:#F00;">Mouse up.</span>');
  }).mousedown(function(){
   $(this).append('<span style="color:#00F;">Mouse down.</span>');
  });
```

mousedown(handler(eventObject))

Bind an event handler to the "mousedown" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mousedown', handler)` in the first variation, and `.trigger('mousedown')` in the second.

The `mousedown` event is sent to an element when the mouse pointer is over the element, and the mouse button is pressed. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$('#target').mousedown(function() {
 alert('Handler for .mousedown() called.');
});
```

Now if we click on this element, the alert is displayed:

Handler for .mousedown() called.

We can also trigger the event when a different element is clicked:

```
$('#other').click(function() {
 $('#target').mousedown();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `mousedown` event is sent when any mouse button is clicked. To act only on specific buttons, we can use the event object's `which` property. Not all browsers support this property (Internet Explorer uses button instead), but jQuery normalizes the property so that it is safe to use in any browser. The value of `which` will be 1 for the left button, 2 for the middle button, or 3 for the right button.

This event is primarily useful for ensuring that the primary button was used to begin a drag operation; if ignored, strange results can occur when the user attempts to use a context menu. While the middle and right buttons can be detected with these properties, this is not reliable. In Opera and Safari, for example, right mouse button clicks are not detectable by default.

If the user clicks on an element, drags away from it, and releases the button, this is still counted as a `mousedown` event. This sequence of actions is treated as a "canceling" of the button press in most user interfaces, so it is usually better to use the `click` event unless we know that the `mousedown` event is preferable for a particular situation.

Example
Show texts when mouseup and mousedown event triggering.

```
$("p").mouseup(function(){
   $(this).append('<span style="color:#F00;">Mouse up.</span>');
  }).mousedown(function(){
   $(this).append('<span style="color:#00F;">Mouse down.</span>');
  });
```

error(handler(eventObject))
Bind an event handler to the "error" JavaScript event.

Arguments
handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for `.bind('error', handler)`.

The `error` event is sent to elements, such as images, that are referenced by a document and loaded by the browser. It is called if the element was not loaded correctly.

For example, consider a page with a simple image element:

```
<img alt="Book" id="book" />
```

The event handler can be bound to the image:

```
$('#book')
 .error(function() {
  alert('Handler for .error() called.')
 })
 .attr("src", "missing.png");
```

If the image cannot be loaded (for example, because it is not present at the supplied URL), the alert is displayed:

Handler for .error() called.

The event handler *must* be attached before the browser fires the error event, which is why the example sets the src attribute after attaching the handler. Also, the error event may not be correctly fired when the page is served locally; `error` relies on HTTP status codes and will generally not be triggered if the URL uses the `file:` protocol.

Note: A jQuery error event handler should not be attached to the window object. The browser fires the window's error event when a script error occurs. However, the window error event receives different arguments and has different return value requirements than conventional event handlers. Use `window.onerror` instead.

Example
To hide the "broken image" icons for IE users, you can try:

```
$("img")
 .error(function(){
```

```
  $(this).hide();
 })
 .attr("src", "missing.png");
```

unload(handler(eventObject))

Bind an event handler to the "unload" JavaScript event.

Arguments

handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for `.bind('unload', handler)`.

The `unload` event is sent to the `window` element when the user navigates away from the page. This could mean one of many things. The user could have clicked on a link to leave the page, or typed in a new URL in the address bar. The forward and back buttons will trigger the event. Closing the browser window will cause the event to be triggered. Even a page reload will first create an `unload` event.

The exact handling of the `unload` event has varied from version to version of browsers. For example, some versions of Firefox trigger the event when a link is followed, but not when the window is closed. In practical usage, behavior should be tested on all supported browsers, and contrasted with the proprietary `beforeunload` event.

Any `unload` event handler should be bound to the `window` object:

```
$(window).unload(function() {
 alert('Handler for .unload() called.');
});
```

After this code executes, the alert will be displayed whenever the browser leaves the current page.It is not possible to cancel the `unload` event with `.preventDefault()`. This event is available so that scripts can perform cleanup when the user leaves the page.

Example

To display an alert when a page is unloaded:

```
$(window).unload( function () { alert("Bye now!"); } );
```

load(handler(eventObject))

Bind an event handler to the "load" JavaScript event.

Arguments

handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for `.bind('load', handler)`.

The `load` event is sent to an element when it and all sub-elements have been completely loaded. This event can be sent to any element associated with a URL: images, scripts, frames, iframes, and the `window` object.

For example, consider a page with a simple image:

```
<img src="book.png" alt="Book" id="book" />
```

The event handler can be bound to the image:

```
$('#book').load(function() {
 // Handler for .load() called.
});
```

As soon as the image has been loaded, the handler is called.

In general, it is not necessary to wait for all images to be fully loaded. If code can be executed earlier, it is usually best to place it in a handler sent to the `.ready()` method.

The Ajax module also has a method named `.load()`. Which one is fired depends on the set of arguments passed.

Caveats of the `load` event when used with images

A common challenge developers attempt to solve using the `.load()` shortcut is to execute a function when an image (or collection of images) have completely loaded. There are several known caveats with this that should be noted. These are:

- It doesn't work consistently nor reliably cross-browser
- It doesn't fire correctly in WebKit if the image src is set to the same src as before
- It doesn't correctly bubble up the DOM tree
- Can cease to fire for images that already live in the browser's cache

Note: The `.live()` and `.delegate()` methods cannot be used to detect the `load` event of an iframe. The load event does not correctly bubble up the parent document and the event.target isn't set by Firefox, IE9 or Chrome, which is required to do event delegation.

Example

Run a function when the page is fully loaded including graphics.

```
$(window).load(function () {
 // run code
});
```

Example

Add the class bigImg to all images with height greater then 100 upon each image load.

```
$('img.userIcon').load(function(){
 if($(this).height() > 100) {
  $(this).addClass('bigImg');
 }
});
```

ready(handler)

Specify a function to execute when the DOM is fully loaded.

Arguments

handler - A function to execute after the DOM is ready.

While JavaScript provides the `load` event for executing code when a page is rendered, this event does not get triggered until all assets such as images have been completely received. In most cases, the script can be run as soon as the DOM hierarchy has been fully constructed. The handler passed to `.ready()` is guaranteed to be executed after the DOM is ready, so this is usually the best place to attach all other event handlers and run other jQuery code. When using scripts that rely on the value of CSS style properties, it's important to reference external stylesheets or embed style elements before referencing the scripts.

In cases where code relies on loaded assets (for example, if the dimensions of an image are required), the code should be placed in a handler for the `load` event instead.

The `.ready()` method is generally incompatible with the `<body onload="">` attribute. If `load` must be used, either do not use `.ready()` or use jQuery's `.load()` method to attach `load` event handlers to the window or to more specific items, like images.

All three of the following syntaxes are equivalent:

- `$(document).ready(handler)`
- `$().ready(handler)` (this is not recommended)
- `$(handler)`

There is also `$(document).bind("ready", handler)`. This behaves similarly to the ready method but with one exception: If the ready event has already fired and you try to `.bind("ready")` the bound handler will not be executed. Ready handlers bound this way are executed *after* any bound by the other three methods above.

The `.ready()` method can only be called on a jQuery object matching the current document, so the selector can be omitted.

The `.ready()` method is typically used with an anonymous function:

```
$(document).ready(function() {
```

```
 // Handler for .ready() called.
});
```

Which is equivalent to calling:

```
$(function() {
 // Handler for .ready() called.
});
```

If `.ready()` is called after the DOM has been initialized, the new handler passed in will be executed immediately.

Aliasing the jQuery Namespace

When using another JavaScript library, we may wish to call `$.noConflict()` to avoid namespace difficulties. When this function is called, the `$` shortcut is no longer available, forcing us to write `jQuery` each time we would normally write `$`. However, the handler passed to the `.ready()` method can take an argument, which is passed the global `jQuery` object. This means we can rename the object within the context of our `.ready()` handler without affecting other code:

```
jQuery(document).ready(function($) {
 // Code using $ as usual goes here.
});
```

Example
Display a message when the DOM is loaded.

```
$(document).ready(function () {
 $("p").text("The DOM is now loaded and can be manipulated.");
});
```

jQuery.browser
Contains flags for the useragent, read from navigator.userAgent. **We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.**

The `$.browser` property provides information about the web browser that is accessing the page, as reported by the browser itself. It contains flags for each of the four most prevalent browser classes (Internet Explorer, Mozilla, Webkit, and Opera) as well as version information.

Available flags are:
- webkit (as of jQuery 1.4)
- safari (deprecated)
- opera
- msie
- mozilla

This property is available immediately. It is therefore safe to use it to determine whether or not to call `$(document).ready()`.The `$.browser` property is deprecated in jQuery 1.3, and its functionality may be moved to a team-supported plugin in a future release of jQuery.

Because `$.browser` uses `navigator.userAgent` to determine the platform, it is vulnerable to spoofing by the user or misrepresentation by the browser itself. It is always best to avoid browser-specific code entirely where possible. The `$.support` property is available for detection of support for particular features rather than relying on `$.browser`.

Example
Show the browser info.

```
jQuery.each(jQuery.browser, function(i, val) {
   $("<div>" + i + " : <span>" + val + "</span>")
        .appendTo( document.body );
  });
```

Example
Returns true if the current useragent is some version of Microsoft's Internet Explorer.

```
$.browser.msie;
```

Example
Alerts "this is WebKit!" only for WebKit browsers

```
if ($.browser.webkit) {
  alert( "this is webkit!" );
 }
```

Example
Alerts "Do stuff for Firefox 3" only for Firefox 3 browsers.

```
var ua = $.browser;
 if ( ua.mozilla && ua.version.slice(0,3) == "1.9" ) {
  alert( "Do stuff for firefox 3" );
 }
```

Example
Set a CSS property that's specific to a particular browser.

```
if ( $.browser.msie ) {
  $("#div ul li").css( "display","inline" );
 } else {
  $("#div ul li").css( "display","inline-table" );
 }
```

jQuery.browser.version
The version number of the rendering engine for the user's browser.

Here are some typical results:
- Internet Explorer: 6.0, 7.0, 8.0
- Mozilla/Firefox/Flock/Camino: 1.7.12, 1.8.1.3, 1.9
- Opera: 10.06, 11.01
- Safari/Webkit: 312.8, 418.9

Note that IE8 claims to be 7 in Compatibility View.

Example
Returns the version number of the rendering engine used by the user's current browser. For example, FireFox 4 returns 2.0 (the version of the Gecko rendering engine it utilizes).

```
$("p").html( "The version number of the rendering engine your browser uses is: <span>" +
        $.browser.version + "</span>" );
```

Example
Alerts the version of IE's rendering engine that is being used:

```
if ( $.browser.msie ) {
 alert( $.browser.version );
}
```

Example
Often you only care about the "major number," the whole number, which you can get by using JavaScript's built-in `parseInt()` function:

```
if ( $.browser.msie ) {
 alert( parseInt($.browser.version, 10) );
}
```

trigger(eventType, [extraParameters])
Execute all handlers and behaviors attached to the matched elements for the given event type.

Arguments

eventType - A string containing a JavaScript event type, such as `click` or `submit`.
extraParameters - Additional parameters to pass along to the event handler.

Any event handlers attached with `.bind()` or one of its shortcut methods are triggered when the corresponding event occurs. They can be fired manually, however, with the `.trigger()` method. A call to `.trigger()` executes the handlers in the same order they would be if the event were triggered naturally by the user:

```
$('#foo').bind('click', function() {
   alert($(this).text());
  });
  $('#foo').trigger('click');
```

As of jQuery 1.3, `.trigger()`ed events bubble up the DOM tree; an event handler can stop the bubbling by returning `false` from the handler or calling the .stopPropagation() method on the event object passed into the event. Although `.trigger()` simulates an event activation, complete with a synthesized event object, it does not perfectly replicate a naturally-occurring event.

To trigger handlers bound via jQuery without also triggering the native event, use .triggerHandler() instead.

When we define a custom event type using the `.bind()` method, the second argument to `.trigger()` can become useful. For example, suppose we have bound a handler for the `custom` event to our element instead of the built-in `click` event as we did above:

```
$('#foo').bind('custom', function(event, param1, param2) {
 alert(param1 + "n" + param2);
});
$('#foo').trigger('custom', ['Custom', 'Event']);
```

The event object is always passed as the first parameter to an event handler, but if additional parameters are specified during a `.trigger()` call, these parameters will be passed along to the handler as well. To pass more than one parameter, use an array as shown here. As of jQuery 1.6.2, a single parameter can be passed without using an array.

Note the difference between the extra parameters we're passing here and the `eventData` parameter to the .bind() method. Both are mechanisms for passing information to an event handler, but the `extraParameters` argument to `.trigger()` allows information to be determined at the time the event is triggered, while the `eventData` argument to `.bind()` requires the information to be already computed at the time the handler is bound.

The `.trigger()` method can be used on jQuery collections that wrap plain JavaScript objects similar to a pub/sub mechanism; any event handlers bound to the object will be called when the event is triggered. **Note:** For both plain objects and DOM objects, if a triggered event name matches the name of a property on the object, jQuery will attempt to invoke the property as a method if no event handler calls `event.preventDefault()`. If this behavior is not desired, use `.triggerHandler()` instead.

Example

Clicks to button #2 also trigger a click for button #1.

```
$("button:first").click(function () {
update($("span:first"));
});
$("button:last").click(function () {
$("button:first").trigger('click');

update($("span:last"));
});

function update(j) {
var n = parseInt(j.text(), 10);
j.text(n + 1);
}
```

Example

To submit the first form without using the submit() function, try:

```
$("form:first").trigger("submit")
```

Example
To submit the first form without using the submit() function, try:

```
var event = jQuery.Event("submit");
$("form:first").trigger(event);
if ( event.isDefaultPrevented() ) {
// Perform an action...
}
```

Example
To pass arbitrary data to an event:

```
$("p").click( function (event, a, b) {
// when a normal click fires, a and b are undefined
// for a trigger like below a refers to "foo" and b refers to "bar"

} ).trigger("click", ["foo", "bar"]);
```

Example
To pass arbitrary data through an event object:

```
var event = jQuery.Event("logged");
event.user = "foo";
event.pass = "bar";
$("body").trigger(event);
```

Example
Alternative way to pass data through an event object:

```
$("body").trigger({
type:"logged",
user:"foo",
pass:"bar"

});
```

ajaxComplete(handler(event, XMLHttpRequest, ajaxOptions))
Register a handler to be called when Ajax requests complete. This is an Ajax Event.

Arguments
handler(event, XMLHttpRequest, ajaxOptions) - The function to be invoked.

Whenever an Ajax request completes, jQuery triggers the `ajaxComplete` event. Any and all handlers that have been registered with the `.ajaxComplete()` method are executed at this time.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxComplete(function() {
 $(this).text('Triggered ajaxComplete handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
```

```
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request completes, the log message is displayed.

Note: Because `.ajaxComplete()` is implemented as a method of jQuery object instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxComplete` handlers are invoked, regardless of what Ajax request was completed. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxComplete` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. For example, we can restrict our callback to only handling events dealing with a particular URL:

Note: You can get the returned ajax contents by looking at `xhr.responseXML` or `xhr.responseHTML` for xml and html respectively.

```
$('.log').ajaxComplete(function(e, xhr, settings) {
 if (settings.url == 'ajax/test.html') {
  $(this).text('Triggered ajaxComplete handler. The result is ' +
           xhr.responseHTML);
 }
});
```

Example

Show a message when an Ajax request completes.

```
$("#msg").ajaxComplete(function(event,request, settings){
  $(this).append("<li>Request Complete.</li>");
 });
```

serialize()

Encode a set of form elements as a string for submission.

The `.serialize()` method creates a text string in standard URL-encoded notation. It operates on a jQuery object representing a set of form elements. The form elements can be of several types:

```
<form>
 <div><input type="text" name="a" value="1" id="a" /></div>
 <div><input type="text" name="b" value="2" id="b" /></div>
 <div><input type="hidden" name="c" value="3" id="c" /></div>
 <div>
  <textarea name="d" rows="8" cols="40">4</textarea>
 </div>
 <div><select name="e">
  <option value="5" selected="selected">5</option>
  <option value="6">6</option>
  <option value="7">7</option>
 </select></div>
 <div>
  <input type="checkbox" name="f" value="8" id="f" />
 </div>
 <div>
  <input type="submit" name="g" value="Submit" id="g" />
 </div>
</form>
```

The `.serialize()` method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization:

```
$('form').submit(function() {
 alert($(this).serialize());
 return false;
```

```
});
```

This produces a standard-looking query string:

```
a=1&b=2&c=3&d=4&e=5
```

Warning: selecting both the form and its children will cause duplicates in the serialized string.

Note: Only "successful controls" are serialized to the string. No submit button value is serialized since the form was not submitted using a button. For a form element's value to be included in the serialized string, the element must have a `name` attribute. Values from checkboxes and radio buttons (`inputs` of type "radio" or "checkbox") are included only if they are checked. Data from file select elements is not serialized.

Example

Serialize a form to a query string, that could be sent to a server in an Ajax request.

```
function showValues() {
   var str = $("form").serialize();
   $("#results").text(str);
  }
  $(":checkbox, :radio").click(showValues);
  $("select").change(showValues);
  showValues();
```

ajaxSuccess(handler(event, XMLHttpRequest, ajaxOptions))

Attach a function to be executed whenever an Ajax request completes successfully. This is an Ajax Event.

Arguments

handler(event, XMLHttpRequest, ajaxOptions) - The function to be invoked.

Whenever an Ajax request completes successfully, jQuery triggers the `ajaxSuccess` event. Any and all handlers that have been registered with the `.ajaxSuccess()` method are executed at this time.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxSuccess(function() {
 $(this).text('Triggered ajaxSuccess handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request completes successfully, the log message is displayed.

Note: Because `.ajaxSuccess()` is implemented as a method of jQuery object instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

All `ajaxSuccess` handlers are invoked, regardless of what Ajax request was completed. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxSuccess` handler is executed, it is passed the event object, the `XMLHttpRequest` object, and the settings object that was used in the creation of the request. For example, we can restrict our callback to only handling events dealing with a particular URL:

Note: You can get the returned ajax contents by looking at `xhr.responseXML` or `xhr.responseText` for xml and html respectively.

```
$('.log').ajaxSuccess(function(e, xhr, settings) {
 if (settings.url == 'ajax/test.html') {
  $(this).text('Triggered ajaxSuccess handler. The ajax response was:'
           + xhr.responseText );
 }
});
```

Example
Show a message when an Ajax request completes successfully.

```
$("#msg").ajaxSuccess(function(evt, request, settings){
   $(this).append("<li>Successful Request!</li>");
   });
```

ajaxStop(handler())
Register a handler to be called when all Ajax requests have completed. This is an Ajax Event.

Arguments
handler() - The function to be invoked.

Whenever an Ajax request completes, jQuery checks whether there are any other outstanding Ajax requests. If none remain, jQuery triggers the `ajaxStop` event. Any and all handlers that have been registered with the `.ajaxStop()` method are executed at this time. The `ajaxStop` event is also triggered if the last outstanding Ajax request is cancelled by returning false within the `beforeSend` callback function.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxStop(function() {
 $(this).text('Triggered ajaxStop handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request completes, the log message is displayed.

Because `.ajaxStop()` is implemented as a method of jQuery object instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

Example
Hide a loading message after all the Ajax requests have stopped.

```
$("#loading").ajaxStop(function(){
   $(this).hide();
});
```

ajaxStart(handler())
Register a handler to be called when the first Ajax request begins. This is an Ajax Event.

Arguments
handler() - The function to be invoked.

Whenever an Ajax request is about to be sent, jQuery checks whether there are any other outstanding Ajax requests. If none are in progress, jQuery

triggers the `ajaxStart` event. Any and all handlers that have been registered with the `.ajaxStart()` method are executed at this time.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxStart(function() {
 $(this).text('Triggered ajaxStart handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request is sent, the log message is displayed.

Note: Because `.ajaxStart()` is implemented as a method of jQuery object instances, we can use the `this` keyword as we do here to refer to the selected elements within the callback function.

Example
Show a loading message whenever an Ajax request starts (and none is already active).

```
$("#loading").ajaxStart(function(){
  $(this).show();
 });
```

ajaxSend(handler(event, jqXHR, ajaxOptions))
Attach a function to be executed before an Ajax request is sent. This is an Ajax Event.

Arguments
handler(event, jqXHR, ajaxOptions) - The function to be invoked.

Whenever an Ajax request is about to be sent, jQuery triggers the `ajaxSend` event. Any and all handlers that have been registered with the `.ajaxSend()` method are executed at this time.

To observe this method in action, we can set up a basic Ajax load request:

```
<div class="trigger">Trigger</div>
<div class="result"></div>
<div class="log"></div>
```

We can attach our event handler to any element:

```
$('.log').ajaxSend(function() {
 $(this).text('Triggered ajaxSend handler.');
});
```

Now, we can make an Ajax request using any jQuery method:

```
$('.trigger').click(function() {
 $('.result').load('ajax/test.html');
});
```

When the user clicks the element with class `trigger` and the Ajax request is about to begin, the log message is displayed.

Note: Because `.ajaxSend()` is implemented as a method of jQuery instances, we can use the `this` keyword as we do here to refer to the selected

elements within the callback function.

All `ajaxSend` handlers are invoked, regardless of what Ajax request is to be sent. If we must differentiate between the requests, we can use the parameters passed to the handler. Each time an `ajaxSend` handler is executed, it is passed the event object, the `jqXHR` object (in version 1.4, `XMLHttpRequest`object), and the settings object that was used in the creation of the Ajax request. For example, we can restrict our callback to only handling events dealing with a particular URL:

```
$('.log').ajaxSend(function(e, jqxhr, settings) {
 if (settings.url == 'ajax/test.html') {
  $(this).text('Triggered ajaxSend handler.');
 }
});
```

Example
Show a message before an Ajax request is sent.

```
$("#msg").ajaxSend(function(evt, request, settings){
    $(this).append("<li>Starting request at " + settings.url + "</li>");
   });
```

ajaxError(handler(event, jqXHR, ajaxSettings, thrownError))
Register a handler to be called when Ajax requests complete with an error. This is an Ajax Event.

Arguments
handler(event, jqXHR, ajaxSettings, thrownError) - The function to be invoked.

Whenever an Ajax request completes with an error, jQuery triggers the `ajaxError` event. Any and all handlers that have been registered with the `.ajaxError()` method are executed at this time.

To observe this method in action, set up a basic Ajax load request.

```
<button class="trigger">Trigger</button>
<div class="result"></div>
<div class="log"></div>
```

Attach the event handler to any element:

```
$("div.log").ajaxError(function() {
 $(this).text( "Triggered ajaxError handler." );
});
```

Now, make an Ajax request using any jQuery method:

```
$("button.trigger").click(function() {
 $("div.result").load( "ajax/missing.html" );
});
```

When the user clicks the button and the Ajax request fails, because the requested file is missing, the log message is displayed.

Note: Because `.ajaxError()` is implemented as a method of jQuery object instances, you can use the `this` keyword within the callback function to refer to the selected elements.

All `ajaxError` handlers are invoked, regardless of what Ajax request was completed. To differentiate between the requests, you can use the parameters passed to the handler. Each time an `ajaxError` handler is executed, it is passed the event object, the `jqXHR` object (prior to jQuery 1.5, the `XHR` object), and the settings object that was used in the creation of the request. If the request failed because JavaScript raised an exception, the exception object is passed to the handler as a fourth parameter. For example, to restrict the error callback to only handling events dealing with a particular URL:

```
$( "div.log" ).ajaxError(function(e, jqxhr, settings, exception) {
 if ( settings.url == "ajax/missing.html" ) {
  $(this).text( "Triggered ajaxError handler." );
 }
```

```
});
```

Example
Show a message when an Ajax request fails.

```
$("#msg").ajaxError(function(event, request, settings){
 $(this).append("<li>Error requesting page " + settings.url + "</li>");
});
```

unbind([eventType], [handler(eventObject)])
Remove a previously-attached event handler from the elements.

Arguments
eventType - A string containing a JavaScript event type, such as `click` or `submit`.
handler(eventObject) - The function that is to be no longer executed.

Event handlers attached with `.bind()` can be removed with `.unbind()`. (As of jQuery 1.7, the .on() and .off() methods are preferred to attach and remove event handlers on elements.) In the simplest case, with no arguments, `.unbind()` removes all handlers attached to the elements:

```
$('#foo').unbind();
```

This version removes the handlers regardless of type. To be more precise, we can pass an event type:

```
$('#foo').unbind('click');
```

By specifying the `click` event type, only handlers for that event type will be unbound. This approach can still have negative ramifications if other scripts might be attaching behaviors to the same element, however. Robust and extensible applications typically demand the two-argument version for this reason:

```
var handler = function() {
 alert('The quick brown fox jumps over the lazy dog.');
};
$('#foo').bind('click', handler);
$('#foo').unbind('click', handler);
```

By naming the handler, we can be assured that no other functions are accidentally removed. Note that the following will *not* work:

```
$('#foo').bind('click', function() {
 alert('The quick brown fox jumps over the lazy dog.');
});

// will NOT work
$('#foo').unbind('click', function() {
 alert('The quick brown fox jumps over the lazy dog.');
});
```

Even though the two functions are identical in content, they are created separately and so JavaScript is free to keep them as distinct function objects. To unbind a particular handler, we need a reference to that function and not a different one that happens to do the same thing.

Note: Using a proxied function to unbind an event on an element will unbind all proxied functions on that element, as the same proxy function is used for all proxied events. To allow unbinding a specific event, use unique class names on the event (e.g. `click.proxy1`, `click.proxy2`) when attaching them.

Using Namespaces

Instead of maintaining references to handlers in order to unbind them, we can namespace the events and use this capability to narrow the scope of our unbinding actions. As shown in the discussion for the `.bind()` method, namespaces are defined by using a period (`.`) character when binding a handler:

```
$('#foo').bind('click.myEvents', handler);
```

When a handler is bound in this fashion, we can still unbind it the normal way:

```
$('#foo').unbind('click');
```

However, if we want to avoid affecting other handlers, we can be more specific:

```
$('#foo').unbind('click.myEvents');
```

We can also unbind all of the handlers in a namespace, regardless of event type:

```
$('#foo').unbind('.myEvents');
```

It is particularly useful to attach namespaces to event bindings when we are developing plug-ins or otherwise writing code that may interact with other event-handling code in the future.

Using the Event Object

The third form of the `.unbind()` method is used when we wish to unbind a handler from within itself. For example, suppose we wish to trigger an event handler only three times:

```
var timesClicked = 0;
$('#foo').bind('click', function(event) {
 alert('The quick brown fox jumps over the lazy dog.');
 timesClicked++;
 if (timesClicked >= 3) {
  $(this).unbind(event);
 }
});
```

The handler in this case must take a parameter, so that we can capture the event object and use it to unbind the handler after the third click. The event object contains the context necessary for `.unbind()` to know which handler to remove.This example is also an illustration of a closure. Since the handler refers to the `timesClicked` variable, which is defined outside the function, incrementing the variable has an effect even between invocations of the handler.

Example

Can bind and unbind events to the colored button.

```
function aClick() {
$("div").show().fadeOut("slow");
}
$("#bind").click(function () {
// could use .bind('click', aClick) instead but for variety...
$("#theone").click(aClick)
 .text("Can Click!");
});
$("#unbind").click(function () {
$("#theone").unbind('click', aClick)
 .text("Does nothing...");
});
```

Example

To unbind all events from all paragraphs, write:

```
$("p").unbind()
```

Example

To unbind all click events from all paragraphs, write:

```
$("p").unbind( "click" )
```

Example

To unbind just one previously bound handler, pass the function in as the second argument:

```
var foo = function () {
// code to handle some kind of event
};

$("p").bind("click", foo); // ... now foo will be called when paragraphs are clicked ...

$("p").unbind("click", foo); // ... foo will no longer be called.
```

bind(eventType, [eventData], handler(eventObject))

Attach a handler to an event for the elements.

Arguments

eventType - A string containing one or more DOM event types, such as "click" or "submit," or custom event names.
eventData - A map of data that will be passed to the event handler.
handler(eventObject) - A function to execute each time the event is triggered.

As of jQuery 1.7, the .on() method is the preferred method for attaching event handlers to a document. For earlier versions, the `.bind()` method is used for attaching an event handler directly to elements. Handlers are attached to the currently selected elements in the jQuery object, so those elements *must exist* at the point the call to `.bind()` occurs. For more flexible event binding, see the discussion of event delegation in .on() or .delegate().

Any string is legal for `eventType`; if the string is not the name of a native DOM event, then the handler is bound to a custom event. These events are never called by the browser, but may be triggered manually from other JavaScript code using `.trigger()` or `.triggerHandler()`.

If the `eventType` string contains a period (`.`) character, then the event is namespaced. The period character separates the event from its namespace. For example, in the call `.bind('click.name', handler)`, the string `click` is the event type, and the string `name` is the namespace. Namespacing allows us to unbind or trigger some events of a type without affecting others. See the discussion of `.unbind()` for more information.

There are shorthand methods for some standard browser events such as .click() that can be used to attach or trigger event handlers. For a complete list of shorthand methods, see the events category.

When an event reaches an element, all handlers bound to that event type for the element are fired. If there are multiple handlers registered, they will always execute in the order in which they were bound. After all handlers have executed, the event continues along the normal event propagation path.

A basic usage of `.bind()` is:

```
$('#foo').bind('click', function() {
 alert('User clicked on "foo."');
});
```

This code will cause the element with an ID of `foo` to respond to the `click` event. When a user clicks inside this element thereafter, the alert will be shown.

Multiple Events

Multiple event types can be bound at once by including each one separated by a space:

```
$('#foo').bind('mouseenter mouseleave', function() {
 $(this).toggleClass('entered');
});
```

The effect of this on `<div id="foo">` (when it does not initially have the "entered" class) is to add the "entered" class when the mouse enters the `<div>` and remove the class when the mouse leaves.

As of jQuery 1.4 we can bind multiple event handlers simultaneously by passing a map of event type/handler pairs:

```
$('#foo').bind({
 click: function() {
  // do something on click
 },
 mouseenter: function() {
  // do something on mouseenter
 }
});
```

Event Handlers

The `handler` parameter takes a callback function, as shown above. Within the handler, the keyword `this` refers to the DOM element to which the handler is bound. To make use of the element in jQuery, it can be passed to the normal `$()` function. For example:

```
$('#foo').bind('click', function() {
 alert($(this).text());
});
```

After this code is executed, when the user clicks inside the element with an ID of `foo`, its text contents will be shown as an alert.

As of jQuery 1.4.2 duplicate event handlers can be bound to an element instead of being discarded. This is useful when the event data feature is being used, or when other unique data resides in a closure around the event handler function.

In jQuery 1.4.3 you can now pass in `false` in place of an event handler. This will bind an event handler equivalent to: `function(){ return false; }`. This function can be removed at a later time by calling: `.unbind( eventName, false )`.

The Event object

The `handler` callback function can also take parameters. When the function is called, the event object will be passed to the first parameter.

The event object is often unnecessary and the parameter omitted, as sufficient context is usually available when the handler is bound to know exactly what needs to be done when the handler is triggered. However, at times it becomes necessary to gather more information about the user's environment at the time the event was initiated. View the full Event Object.

Returning `false` from a handler is equivalent to calling both `.preventDefault()` and `.stopPropagation()` on the event object.

Using the event object in a handler looks like this:

```
$(document).ready(function() {
 $('#foo').bind('click', function(event) {
  alert('The mouse cursor is at ('
   + event.pageX + ', ' + event.pageY + ')');
 });
});
```

Note the parameter added to the anonymous function. This code will cause a click on the element with ID `foo` to report the page coordinates of the mouse cursor at the time of the click.

Passing Event Data

The optional `eventData` parameter is not commonly used. When provided, this argument allows us to pass additional information to the handler. One handy use of this parameter is to work around issues caused by closures. For example, suppose we have two event handlers that both refer to the same external variable:

```
var message = 'Spoon!';
$('#foo').bind('click', function() {
```

```
 alert(message);
});
message = 'Not in the face!';
$('#bar').bind('click', function() {
 alert(message);
});
```

Because the handlers are closures that both have `message` in their environment, both will display the message Not in the face! when triggered. The variable's value has changed. To sidestep this, we can pass the message in using `eventData`:

```
var message = 'Spoon!';
$('#foo').bind('click', {msg: message}, function(event) {
 alert(event.data.msg);
});
message = 'Not in the face!';
$('#bar').bind('click', {msg: message}, function(event) {
 alert(event.data.msg);
});
```

This time the variable is not referred to directly within the handlers; instead, the variable is passed in *by value* through `eventData`, which fixes the value at the time the event is bound. The first handler will now display Spoon! while the second will alert Not in the face!

Note that objects are passed to functions *by reference*, which further complicates this scenario.

If `eventData` is present, it is the second argument to the `.bind()` method; if no additional data needs to be sent to the handler, then the callback is passed as the second and final argument.

See the `.trigger()` method reference for a way to pass data to a handler at the time the event happens rather than when the handler is bound.

As of jQuery 1.4 we can no longer attach data (and thus, events) to object, embed, or applet elements because critical errors occur when attaching data to Java applets.

Note: Although demonstrated in the next example, it is inadvisable to bind handlers to both the `click` and `dblclick` events for the same element. The sequence of events triggered varies from browser to browser, with some receiving two click events before the `dblclick` and others only one. Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

Example

Handle click and double-click for the paragraph. Note: the coordinates are window relative, so in this case relative to the demo iframe.

```
$("p").bind("click", function(event){
var str = "( " + event.pageX + ", " + event.pageY + " )";
$("span").text("Click happened! " + str);
});
$("p").bind("dblclick", function(){
$("span").text("Double-click happened in " + this.nodeName);
});
$("p").bind("mouseenter mouseleave", function(event){
$(this).toggleClass("over");
});
```

Example

To display each paragraph's text in an alert box whenever it is clicked:

```
$("p").bind("click", function(){
alert( $(this).text() );
});
```

Example

You can pass some extra data before the event handler:

```
function handler(event) {
alert(event.data.foo);
}
$("p").bind("click", {foo: "bar"}, handler)
```

Example
Cancel a default action and prevent it from bubbling up by returning `false`:

```
$("form").bind("submit", function() { return false; })
```

Example
Cancel only the default action by using the .preventDefault() method.

```
$("form").bind("submit", function(event) {
event.preventDefault();
});
```

Example
Stop an event from bubbling without preventing the default action by using the .stopPropagation() method.

```
$("form").bind("submit", function(event) {
 event.stopPropagation();
});
```

Example
Bind custom events.

```
$("p").bind("myCustomEvent", function(e, myName, myValue){
$(this).text(myName + ", hi there!");
$("span").stop().css("opacity", 1)
.text("myName = " + myName)
.fadeIn(30).fadeOut(1000);
});
$("button").click(function () {
$("p").trigger("myCustomEvent", [ "John" ]);
});
```

Example
Bind multiple events simultaneously.

```
$("div.test").bind({
 click: function(){
  $(this).addClass("active");
 },
 mouseenter: function(){
  $(this).addClass("inside");
 },
 mouseleave: function(){
  $(this).removeClass("inside");
 }
});
```

jQuery(selector, [context])
Accepts a string containing a CSS selector which is then used to match a set of elements.

Arguments
selector - A string containing a selector expression
context - A DOM Element, Document, or jQuery to use as context

In the first formulation listed above, `jQuery()` - which can also be written as `$()` - searches through the DOM for any elements that match the provided selector and creates a new jQuery object that references these elements:

```
$('div.foo');
```

If no elements match the provided selector, the new jQuery object is "empty"; that is, it contains no elements and has `.length` property of 0.

Selector Context

By default, selectors perform their searches within the DOM starting at the document root. However, an alternate context can be given for the search by using the optional second parameter to the `$()` function. For example, to do a search within an event handler, the search can be restricted like so:

```
$('div.foo').click(function() {
 $('span', this).addClass('bar');
});
```

When the search for the span selector is restricted to the context of `this`, only spans within the clicked element will get the additional class.

Internally, selector context is implemented with the `.find()` method, so `$('span', this)` is equivalent to `$(this).find('span')`.

Using DOM elements

The second and third formulations of this function create a jQuery object using one or more DOM elements that were already selected in some other way. A common use of this facility is to call jQuery methods on an element that has been passed to a callback function through the keyword `this`:

```
$('div.foo').click(function() {
 $(this).slideUp();
});
```

This example causes elements to be hidden with a sliding animation when clicked. Because the handler receives the clicked item in the `this` keyword as a bare DOM element, the element must be passed to the `$()` function before applying jQuery methods to it.

XML data returned from an Ajax call can be passed to the `$()` function so individual elements of the XML structure can be retrieved using `.find()` and other DOM traversal methods.

```
$.post('url.xml', function(data) {
 var $child = $(data).find('child');
})
```

Cloning jQuery Objects

When a jQuery object is passed to the `$()` function, a clone of the object is created. This new jQuery object references the same DOM elements as the initial one.

Returning an Empty Set

As of jQuery 1.4, calling the `jQuery()` method with *no arguments* returns an empty jQuery set (with a `.length` property of 0). In previous versions of jQuery, this would return a set containing the document node.

Working With Plain Objects

At present, the only operations supported on plain JavaScript objects wrapped in jQuery are: `.data()`,`.prop()`,`.bind()`, `.unbind()`, `.trigger()` and `.triggerHandler()`. The use of `.data()` (or any method requiring `.data()`) on a plain object will result in a new property on the object called jQuery{randomNumber} (eg. jQuery123456789).

```
// define a plain object
var foo = {foo:'bar', hello:'world'};

// wrap this with jQuery
var $foo = $(foo);

// test accessing property values
var test1 = $foo.prop('foo'); // bar

// test setting property values
$foo.prop('foo', 'foobar');
var test2 = $foo.prop('foo'); // foobar

// test using .data() as summarized above
$foo.data('keyName', 'someValue');
console.log($foo); // will now contain a jQuery{randomNumber} property

// test binding an event name and triggering
$foo.bind('eventName', function (){
    console.log('eventName was called');
});

$foo.trigger('eventName'); // logs 'eventName was called'
```

Should `.trigger('eventName')` be used, it will search for an 'eventName' property on the object and attempt to execute it after any attached jQuery handlers are executed. It does not check whether the property is a function or not. To avoid this behavior, `.triggerHandler('eventName')` should be used instead.

```
$foo.triggerHandler('eventName'); // also logs 'eventName was called'
```

Example
Find all p elements that are children of a div element and apply a border to them.

```
$("div > p").css("border", "1px solid gray");
```

Example
Find all inputs of type radio within the first form in the document.

```
$("input:radio", document.forms[0]);
```

Example
Find all div elements within an XML document from an Ajax response.

```
$("div", xml.responseXML);
```

Example
Set the background color of the page to black.

```
$(document.body).css( "background", "black" );
```

Example
Hide all the input elements within a form.

```
$(myForm.elements).hide()
```

jQuery(html, [ownerDocument])
Creates DOM elements on the fly from the provided string of raw HTML.

Arguments

html - A string of HTML to create on the fly. Note that this parses HTML, **not** XML.
ownerDocument - A document in which the new elements will be created

Creating New Elements

If a string is passed as the parameter to `$()`, jQuery examines the string to see if it looks like HTML (i.e., it has `<tag ... >` somewhere within the string). If not, the string is interpreted as a selector expression, as explained above. But if the string appears to be an HTML snippet, jQuery attempts to create new DOM elements as described by the HTML. Then a jQuery object is created and returned that refers to these elements. You can perform any of the usual jQuery methods on this object:

```
$('<p id="test">My <em>new</em> text</p>').appendTo('body');
```

If the HTML is more complex than a single tag without attributes, as it is in the above example, the actual creation of the elements is handled by the browser's `innerHTML` mechanism. In most cases, jQuery creates a new element and sets the innerHTML property of the element to the HTML snippet that was passed in. When the parameter has a single tag, such as `$('<img />')` or `$('<a></a>')`, jQuery creates the element using the native JavaScript `createElement()` function.

When passing in complex HTML, some browsers may not generate a DOM that exactly replicates the HTML source provided. As mentioned, we use the browser's `.innerHTML` property to parse the passed HTML and insert it into the current document. During this process, some browsers filter out certain elements such as `<html>`, `<title>`, or `<head>` elements. As a result, the elements inserted may not be representative of the original string passed.

Filtering isn't however just limited to these tags. For example, Internet Explorer prior to version 8 will also convert all `href` properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate compatibility layer.

To ensure cross-platform compatibility, the snippet must be well-formed. Tags that can contain other elements should be paired with a closing tag:

```
$('<a href="http://jquery.com"></a>');
```

Alternatively, jQuery allows XML-like tag syntax (with or without a space before the slash):

```
$('<a/>');
```

Tags that cannot contain elements may be quick-closed or not:

```
$('<img />');
$('<input>');
```

When passing HTML to `jQuery()`, please also note that text nodes are not treated as DOM elements. With the exception of a few methods (such as `.content()`), they are generally otherwise ignored or removed. E.g:

```
var el = $('1<br/>2<br/>3'); // returns [<br>, "2", <br>]
el = $('1<br/>2<br/>3 >'); // returns [<br>, "2", <br>, "3 &gt;"]
```

This behaviour is expected.

As of jQuery 1.4, the second argument to `jQuery()` can accept a map consisting of a superset of the properties that can be passed to the .attr() method. Furthermore, any event type can be passed in, and the following jQuery methods can be called: val, css, html, text, data, width, height, or offset. The name `"class"` must be quoted in the map since it is a JavaScript reserved word, and `"className"` cannot be used since it is not the correct attribute name.

Note: Internet Explorer will not allow you to create an `input` or `button` element and change its type; you must specify the type using `'<input type="checkbox" />'` for example. A demonstration of this can be seen below:

Unsupported in IE:

```
$('<input />', {
```

```
  type: 'text',
  name: 'test'
}).appendTo("body");
```

Supported workaround:

```
$('<input type="text" />').attr({
  name: 'test'
}).appendTo("body");
```

Example
Create a div element (and all of its contents) dynamically and append it to the body element. Internally, an element is created and its innerHTML property set to the given markup.

```
$("<div><p>Hello</p></div>").appendTo("body")
```

Example
Create some DOM elements.

```
$("<div/>", {
 "class": "test",
 text: "Click me!",
 click: function(){
  $(this).toggleClass("test");
 }
}).appendTo("body");
```

jQuery(callback)
Binds a function to be executed when the DOM has finished loading.

Arguments
callback - The function to execute when the DOM is ready.

This function behaves just like `$(document).ready()`, in that it should be used to wrap other `$()` operations on your page that depend on the DOM being ready. While this function is, technically, chainable, there really isn't much use for chaining against it.

Example
Execute the function when the DOM is ready to be used.

```
$(function(){
  // Document is ready
 });
```

Example
Use both the shortcut for $(document).ready() and the argument to write failsafe jQuery code using the $ alias, without relying on the global alias.

```
jQuery(function($) {
  // Your code using failsafe $ alias here...
 });
```

end()
End the most recent filtering operation in the current chain and return the set of matched elements to its previous state.

Most of jQuery's DOM traversal methods operate on a jQuery object instance and produce a new one, matching a different set of DOM elements. When this happens, it is as if the new set of elements is pushed onto a stack that is maintained inside the object. Each successive filtering method pushes a new element set onto the stack. If we need an older element set, we can use `end()` to pop the sets back off of the stack.

Suppose we have a couple short lists on a page:

```
<ul class="first">
  <li class="foo">list item 1</li>
  <li>list item 2</li>
  <li class="bar">list item 3</li>
</ul>
<ul class="second">
  <li class="foo">list item 1</li>
  <li>list item 2</li>
  <li class="bar">list item 3</li>
</ul>
```

The `end()` method is useful primarily when exploiting jQuery's chaining properties. When not using chaining, we can usually just call up a previous object by variable name, so we don't need to manipulate the stack. With `end()`, though, we can string all the method calls together:

```
$('ul.first').find('.foo').css('background-color', 'red')
 .end().find('.bar').css('background-color', 'green');
```

This chain searches for items with the class `foo` within the first list only and turns their backgrounds red. Then `end()` returns the object to its state before the call to `find()`, so the second `find()` looks for '.bar' inside `<ul class="first">`, not just inside that list's `<li class="foo">`, and turns the matching elements' backgrounds green. The net result is that items 1 and 3 of the first list have a colored background, and none of the items from the second list do.

A long jQuery chain can be visualized as a structured code block, with filtering methods providing the openings of nested blocks and `end()` methods closing them:

```
$('ul.first').find('.foo')
 .css('background-color', 'red')
.end().find('.bar')
 .css('background-color', 'green')
.end();
```

The last `end()` is unnecessary, as we are discarding the jQuery object immediately thereafter. However, when the code is written in this form, the `end()` provides visual symmetry and a sense of completion -making the program, at least to the eyes of some developers, more readable, at the cost of a slight hit to performance as it is an additional function call.

Example

Selects all paragraphs, finds span elements inside these, and reverts the selection back to the paragraphs.

```
jQuery.fn.showTags = function (n) {
   var tags = this.map(function () {
               return this.tagName;
             })
           .get().join(", ");
   $("b:eq(" + n + ")").text(tags);
   return this;
  };

  $("p").showTags(0)
     .find("span")
     .showTags(1)
     .css("background", "yellow")
     .end()
     .showTags(2)
     .css("font-style", "italic");
```

Example
Selects all paragraphs, finds span elements inside these, and reverts the selection back to the paragraphs.

```
$("p").find("span").end().css("border", "2px red solid");
```

siblings([selector])
Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.siblings()` method allows us to search through the siblings of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find its siblings:

```
$('li.third-item').siblings().css('background-color', 'red');
```

The result of this call is a red background behind items 1, 2, 4, and 5. Since we do not supply a selector expression, all of the siblings are part of the object. If we had supplied one, only the matching items among these four would be included.

The original element is not included among the siblings, which is important to remember when we wish to find all elements at a particular level of the DOM tree.

Example
Find the unique siblings of all yellow li elements in the 3 lists (including other yellow li elements if appropriate).

```
var len = $(".hilite").siblings()
             .css("color", "red")
             .length;
  $("b").text(len);
```

Example
Find all siblings with a class "selected" of each div.

```
$("p").siblings(".selected").css("background", "yellow");
```

animate(properties, [duration], [easing], [complete])
Perform a custom animation of a set of CSS properties.

Arguments
properties - A map of CSS properties that the animation will move toward.
duration - A string or number determining how long the animation will run.
easing - A string indicating which easing function to use for the transition.
complete - A function to call once the animation is complete.

The `.animate()` method allows us to create animation effects on any numeric CSS property. The only required parameter is a map of CSS properties. This map is similar to the one that can be sent to the `.css()` method, except that the range of properties is more restrictive.

Animation Properties and Values

All animated properties should be animated to a *single numeric value*, except as noted below; most properties that are non-numeric cannot be animated using basic jQuery functionality (For example, `width`, `height`, or `left` can be animated but `background-color` cannot be, unless the jQuery.Color() plugin is used). Property values are treated as a number of pixels unless otherwise specified. The units `em` and `%` can be specified where applicable.

In addition to style properties, some non-style properties such as `scrollTop` and `scrollLeft`, as well as custom properties, can be animated.

Shorthand CSS properties (e.g. font, background, border) are not fully supported. For example, if you want to animate the rendered border width, at least a border style and border width other than "auto" must be set in advance. Or, if you want to animate font size, you would use `fontSize` or the CSS equivalent `'font-size'` rather than simply `'font'`.

In addition to numeric values, each property can take the strings `'show'`, `'hide'`, and `'toggle'`. These shortcuts allow for custom hiding and showing animations that take into account the display type of the element.

Animated properties can also be relative. If a value is supplied with a leading `+=` or `-=` sequence of characters, then the target value is computed by adding or subtracting the given number from the current value of the property.

Note: Unlike shorthand animation methods such as `.slideDown()` and `.fadeIn()`, the `.animate()` method does *not* make hidden elements visible as part of the effect. For example, given `$('someElement').hide().animate({height:'20px'}, 500)`, the animation will run, but *the element will remain hidden*.

Duration

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The default duration is `400` milliseconds. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Complete Function

If supplied, the `complete` callback function is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, the callback is executed once per matched element, not once for the animation as a whole.

Basic Usage

To animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123"
 style="position: relative; left: 10px;" />
```

To animate the opacity, left offset, and height of the image simultaneously:

```
$('#clickme').click(function() {
 $('#book').animate({
  opacity: 0.25,
  left: '+=50',
  height: 'toggle'
 }, 5000, function() {
  // Animation complete.
 });
});
```

Note that the target value of the `height` property is `'toggle'`. Since the image was visible before, the animation shrinks the height to 0 to hide it. A second click then reverses this transition:

The `opacity` of the image is already at its target value, so this property is not animated by the second click. Since the target value for `left` is a relative value, the image moves even farther to the right during this second animation.

Directional properties (`top`, `right`, `bottom`, `left`) have no discernible effect on elements if their `position` style property is `static`, which it is by default.

Note: The jQuery UI project extends the `.animate()` method by allowing some non-numeric styles such as colors to be animated. The project also includes mechanisms for specifying animations through CSS classes rather than individual attributes.

Note: if attempting to animate an element with a height or width of 0px, where contents of the element are visible due to overflow, jQuery may clip this overflow during animation. By fixing the dimensions of the original element being hidden however, it is possible to ensure that the animation runs smoothly. A clearfix can be used to automatically fix the dimensions of your main element without the need to set this manually.

Step Function

The second version of `.animate()` provides a `step` option - a callback function that is fired at each step of the animation. This function is useful for enabling custom animation types or altering the animation as it is occurring. It accepts two arguments (`now` and `fx`), and `this` is set to the DOM element being animated.

- now: the numeric value of the property being animated at each step
- `fx`: a reference to the `jQuery.fx` prototype object, which contains a number of properties such as `elem` for the animated element, `start` and `end` for the first and last value of the animated property, respectively, and `prop` for the property being animated.

Note that the `step` function is called for each animated property on each animated element. For example, given two list items, the `step` function fires four times at each step of the animation:

```
$('li').animate({
 opacity: .5,
 height: '50%'
},
{
 step: function(now, fx) {
  var data = fx.elem.id + ' ' + fx.prop + ': ' + now;
  $('body').append('<div>' + data + '</div>');
 }
});
```

Easing

The remaining parameter of `.animate()` is a string naming an easing function to use. An easing function specifies the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Per-property Easing

As of jQuery version 1.4, you can set per-property easing functions within a single `.animate()` call. In the first version of `.animate()`, each property can take an array as its value: The first member of the array is the CSS property and the second member is an easing function. If a per-property easing function is not defined for a particular property, it uses the value of the `.animate()` method's optional easing argument. If the easing argument is not defined, the default `swing` function is used.

For example, to simultaneously animate the width and height with the `swing` easing function and the opacity with the `linear` easing function:

```
$('#clickme').click(function() {
 $('#book').animate({
  width: ['toggle', 'swing'],
  height: ['toggle', 'swing'],
```

```
  opacity: 'toggle'
 }, 5000, 'linear', function() {
   $(this).after('<div>Animation complete.</div>');
 });
});
```

In the second version of `.animate()`, the options map can include the `specialEasing` property, which is itself a map of CSS properties and their corresponding easing functions. For example, to simultaneously animate the width using the `linear` easing function and the height using the `easeOutBounce` easing function:

```
$('#clickme').click(function() {
 $('#book').animate({
  width: 'toggle',
  height: 'toggle'
 }, {
  duration: 5000,
  specialEasing: {
   width: 'linear',
   height: 'easeOutBounce'
  },
  complete: function() {
   $(this).after('<div>Animation complete.</div>');
  }
 });
});
```

As previously noted, a plugin is required for the `easeOutBounce` function.

Example

Click the button to animate the div with a number of different properties.

```
/* Using multiple unit types within one animation. */

$("#go").click(function(){
 $("#block").animate({
  width: "70%",
  opacity: 0.4,
  marginLeft: "0.6in",
  fontSize: "3em",
  borderWidth: "10px"
 }, 1500 );
});
```

Example

Animates a div's left property with a relative value. Click several times on the buttons to see the relative animations queued up.

```
$("#right").click(function(){
 $(".block").animate({"left": "+=50px"}, "slow");
});

$("#left").click(function(){
 $(".block").animate({"left": "-=50px"}, "slow");
});
```

Example

The first button shows how an unqueued animation works. It expands the div out to 90% width **while** the font-size is increasing. Once the font-size change is complete, the border animation will begin.The second button starts a traditional chained animation, where each animation will start once the previous animation on the element has completed.

```
$( "#go1" ).click(function(){
 $( "#block1" ).animate( { width: "90%" }, { queue: false, duration: 3000 })
   .animate({ fontSize: "24px" }, 1500 )
   .animate({ borderRightWidth: "15px" }, 1500 );
```

```
});

$( "#go2" ).click(function(){
 $( "#block2" ).animate({ width: "90%" }, 1000 )
   .animate({ fontSize: "24px" }, 1000 )
   .animate({ borderLeftWidth: "15px" }, 1000 );
});

$( "#go3" ).click(function(){
 $( "#go1" ).add( "#go2" ).click();
});

$( "#go4" ).click(function(){
 $( "div" ).css({ width: "", fontSize: "", borderWidth: "" });
});
```

Example

Animates the first div's left property and synchronizes the remaining divs, using the step function to set their left properties at each stage of the animation.

```
$( "#go" ).click(function(){
 $( ".block:first" ).animate({
  left: 100
 }, {
  duration: 1000,
  step: function( now, fx ){
   $( ".block:gt(0)" ).css( "left", now );
  }
 });
});
```

Example

Animate all paragraphs to toggle both height and opacity, completing the animation within 600 milliseconds.

```
$( "p" ).animate({
 height: "toggle", opacity: "toggle"
}, "slow" );
```

Example

Animate all paragraphs to a left style of 50 and opacity of 1 (opaque, visible), completing the animation within 500 milliseconds.

```
$( "p" ).animate({
 left: 50, opacity: 1
}, 500 );
```

Example

Animate the left and opacity style properties of all paragraphs; run the animation *outside* the queue, so that it will automatically start without waiting for its turn.

```
$( "p" ).animate({
 left: "50px", opacity: 1
}, { duration: 500, queue: false });
```

Example

An example of using an 'easing' function to provide a different style of animation. This will only work if you have a plugin that provides this easing function. Note, this code will do nothing unless the paragraph element is hidden.

```
$( "p" ).animate({
 opacity: "show"
}, "slow", "easein" );
```

Example

Animates all paragraphs to toggle both height and opacity, completing the animation within 600 milliseconds.

```
$( "p" ).animate({
 height: "toggle", opacity: "toggle"
}, { duration: "slow" });
```

Example
Use an easing function to provide a different style of animation. This will only work if you have a plugin that provides this easing function.

```
$( "p" ).animate({
 opacity: "show"
}, { duration: "slow", easing: "easein" });
```

Example
Animate all paragraphs and execute a callback function when the animation is complete. The first argument is a map of CSS properties, the second specifies that the animation should take 1000 milliseconds to complete, the third states the easing type, and the fourth argument is an anonymous callback function.

```
$( "p" ).animate({
 height: 200, width: 400, opacity: 0.5
}, 1000, "linear", function() {
 alert("all done");
});
```

prev([selector])
Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.prev()` method searches for the predecessor of each of these elements in the DOM tree and constructs a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that can be passed to the `$()` function. If the selector is supplied, the preceding element will be filtered by testing whether it match the selector.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

To select the element that comes immediately before item three:

```
$('li.third-item').prev().css('background-color', 'red');
```

The result of this call is a red background behind item 2. Since no selector expression is supplied, this preceding element is unequivocally included as part of the object. If one had been supplied, the element would be tested for a match before it was included.

If no previous sibling exists, or if the previous sibling element does not match a supplied selector, an empty jQuery object is returned.

To select *all* preceding sibling elements, rather than just the preceding *adjacent* sibling, use the .prevAll() method.

Example
Find the very previous sibling of each div.

```
var $curr = $("#start");
  $curr.css("background", "#f99");
```

```
  $("button").click(function () {
   $curr = $curr.prev();
   $("div").css("background", "");
   $curr.css("background", "#f99");
  });
```

Example
For each paragraph, find the very previous sibling that has a class "selected".

```
$("p").prev(".selected").css("background", "yellow");
```

fadeTo(duration, opacity, [callback])
Adjust the opacity of the matched elements.

Arguments
duration - A string or number determining how long the animation will run.
opacity - A number between 0 and 1 denoting the target opacity.
callback - A function to call once the animation is complete.

The `.fadeTo()` method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, the default duration of `400` milliseconds is used. Unlike the other effect methods, `.fadeTo()` requires that `duration` be explicitly specified.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
  Click here
 </div>
 <img id="book" src="book.png" alt="" width="100" height="123" />
 With the element initially shown, we can dim it slowly:
 $('#clickme').click(function() {
  $('#book').fadeTo('slow', 0.5, function() {
   // Animation complete.
  });
 });
```

With `duration` set to `0`, this method just changes the `opacity` CSS property, so `.fadeTo(0, opacity)` is the same as `.css('opacity', opacity)`.

Example
Animates first paragraph to fade to an opacity of 0.33 (33%, about one third visible), completing the animation within 600 milliseconds.

```
$("p:first").click(function () {
$(this).fadeTo("slow", 0.33);
});
```

Example
Fade div to a random opacity on each click, completing the animation within 200 milliseconds.

```
$("div").click(function () {
$(this).fadeTo("fast", Math.random());
});
```

Example
Find the right answer! The fade will take 250 milliseconds and change various styles when it completes.

```
var getPos = function (n) {
return (Math.floor(n) * 90) + "px";
};
$("p").each(function (n) {
var r = Math.floor(Math.random() * 3);
var tmp = $(this).text();
$(this).text($("p:eq(" + r + ")").text());
$("p:eq(" + r + ")").text(tmp);
$(this).css("left", getPos(n));
});
$("div").each(function (n) {
   $(this).css("left", getPos(n));
  })
.css("cursor", "pointer")
.click(function () {
   $(this).fadeTo(250, 0.25, function () {
      $(this).css("cursor", "")
          .prev().css({"font-weight": "bolder",
                "font-style": "italic"});
     });
  });
```

fadeOut([duration], [callback])
Hide the matched elements by fading them to transparent.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.fadeOut()` method animates the opacity of the matched elements. Once the opacity reaches 0, the `display` style property is set to `none`, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially shown, we can hide it slowly:

```
$('#clickme').click(function() {
 $('#book').fadeOut('slow', function() {
  // Animation complete.
 });
});
```

Note: To avoid unnecessary DOM manipulation, `.fadeOut()` will not hide an element that is already considered hidden. For information on which elements jQuery considers hidden, see :hidden Selector.

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example
Animates all paragraphs to fade out, completing the animation within 600 milliseconds.

```
$("p").click(function () {
 $("p").fadeOut("slow");
 });
```

Example
Fades out spans in one section that you click on.

```
$("span").click(function () {
 $(this).fadeOut(1000, function () {
 $("div").text("'" + $(this).text() + "' has faded!");
 $(this).remove();
 });
 });
 $("span").hover(function () {
 $(this).addClass("hilite");
 }, function () {
 $(this).removeClass("hilite");
 });
```

Example
Fades out two divs, one with a "linear" easing and one with the default, "swing," easing.

```
$("#btn1").click(function() {
 function complete() {
  $("<div/>").text(this.id).appendTo("#log");
 }

 $("#box1").fadeOut(1600, "linear", complete);
 $("#box2").fadeOut(1600, complete);
});

$("#btn2").click(function() {
 $("div").show();
 $("#log").empty();
});
```

parents([selector])
Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.parents()` method allows us to search through the ancestors of these elements in the DOM tree and construct a new jQuery object from the matching elements ordered from immediate parent on up; the elements are returned in order from the closest parent to the outer ones. The `.parents()` and `.parent()` methods are similar, except that the latter only travels a single

level up the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at item A, we can find its ancestors:

```
$('li.item-a').parents().css('background-color', 'red');
```

The result of this call is a red background for the level-2 list, item II, and the level-1 list (and on up the DOM tree all the way to the `<html>` element). Since we do not supply a selector expression, all of the ancestors are part of the returned jQuery object. If we had supplied one, only the matching items among these would be included.

Example

Find all parent elements of each b.

```
var parentEls = $("b").parents()
      .map(function () {
         return this.tagName;
        })
      .get().join(", ");
$("b").append("<strong>" + parentEls + "</strong>");
```

Example

Click to find all unique div parent elements of each span.

```
function showParents() {
 $("div").css("border-color", "white");
 var len = $("span.selected")
          .parents("div")
          .css("border", "2px red solid")
          .length;
 $("b").text("Unique div parents: " + len);
}
$("span").click(function () {
 $(this).toggleClass("selected");
 showParents();
});
```

fadeIn([duration], [callback])

Display the matched elements by fading them to opaque.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.fadeIn()` method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
   Click here
  </div>
  <img id="book" src="book.png" alt="" width="100" height="123" />
  With the element initially hidden, we can show it slowly:
  $('#clickme').click(function() {
   $('#book').fadeIn('slow', function() {
    // Animation complete
   });
  });
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example
Animates hidden divs to fade in one by one, completing each animation within 600 milliseconds.

```
$(document.body).click(function () {
    $("div:hidden:first").fadeIn("slow");
   });
```

Example
Fades a red block in over the text. Once the animation is done, it quickly fades in more text on top.

```
$("a").click(function () {
     $("div").fadeIn(3000, function () {
      $("span").fadeIn(100);
     });
     return false;
    });
```

parent([selector])

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.parent()` method allows us to search through the parents of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.parents()` and `.parent()` methods are similar, except that the latter only travels a single level up the DOM tree.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at item A, we can find its parents:

```
$('li.item-a').parent().css('background-color', 'red');
```

The result of this call is a red background for the level-2 list. Since we do not supply a selector expression, the parent element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

Example
Shows the parent of each element as (parent > child). Check the View Source to see the raw html.

```
$("*", document.body).each(function () {
   var parentTag = $(this).parent().get(0).tagName;
   $(this).prepend(document.createTextNode(parentTag + " > "));
  });
```

Example
Find the parent element of each paragraph with a class "selected".

```
$("p").parent(".selected").css("background", "yellow");
```

slideToggle([duration], [callback])
Display or hide the matched elements with a sliding motion.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideToggle()` method animates the height of the matched elements. This causes lower parts of the page to slide up or down, appearing to

reveal or conceal the items. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`. When the height reaches 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

We will cause `.slideToggle()` to be called when another element is clicked:

```
$('#clickme').click(function() {
 $('#book').slideToggle('slow', function() {
  // Animation complete.
 });
});
```

With the element initially shown, we can hide it slowly with the first click:

A second click will show the element once again:

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all paragraphs to slide up or down, completing the animation within 600 milliseconds.

```
$("button").click(function () {
   $("p").slideToggle("slow");
  });
```

Example

Animates divs between dividers with a toggle that makes some appear and some disappear.

```
$("#aa").click(function () {
  $("div:not(.still)").slideToggle("slow", function () {
   var n = parseInt($("span").text(), 10);
   $("span").text(n + 1);
```

```
  });
 });
```

jQuery.post(url, [data], [success(data, textStatus, jqXHR)], [dataType])

Load data from the server using a HTTP POST request.

Arguments

url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.
dataType - The type of data expected from the server. Default: Intelligent Guess (xml, json, script, text, html).

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 type: 'POST',
 url: url,
 data: data,
 success: success,
 dataType: dataType
});
```

The `success` callback function is passed the returned data, which will be an XML root element or a text string depending on the MIME type of the response. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function is also passed a "jqXHR" object (in **jQuery 1.4**, it was passed the `XMLHttpRequest` object).

Most implementations will specify a success handler:

```
$.post('ajax/test.html', function(data) {
 $('.result').html(data);
});
```

This example fetches the requested HTML snippet and inserts it on the page.

Pages fetched with `POST` are never cached, so the `cache` and `ifModified` options in `jQuery.ajaxSetup()` have no effect on these requests.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.post()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.post()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
  // and remember the jqxhr object for this request
  var jqxhr = $.post("example.php", function() {
   alert("success");
  })
  .success(function() { alert("second success"); })
  .error(function() { alert("error"); })
  .complete(function() { alert("complete"); });

  // perform other work here ...
```

```
  // Set another completion function for the request above
  jqxhr.complete(function(){ alert("second complete"); });
```

Example
Request the test.php page, but ignore the return results.

```
$.post("test.php");
```

Example
Request the test.php page and send some additional data along (while still ignoring the return results).

```
$.post("test.php", { name: "John", time: "2pm" } );
```

Example
pass arrays of data to the server (while still ignoring the return results).

```
$.post("test.php", { 'choices[]': ["Jon", "Susan"] });
```

Example
send form data using ajax requests

```
$.post("test.php", $("#testform").serialize());
```

Example
Alert out the results from requesting test.php (HTML or XML, depending on what was returned).

```
$.post("test.php", function(data) {
  alert("Data Loaded: " + data);
 });
```

Example
Alert out the results from requesting test.php with an additional payload of data (HTML or XML, depending on what was returned).

```
$.post("test.php", { name: "John", time: "2pm" },
  function(data) {
   alert("Data Loaded: " + data);
  });
```

Example
Gets the test.php page content, store it in a XMLHttpResponse object and applies the process() JavaScript function.

```
$.post("test.php", { name: "John", time: "2pm" },
 function(data) {
  process(data);
 },
 "xml"
);
```

Example
Posts to the test.php page and gets contents which has been returned in json format (<?php echo json_encode(array("name"=>"John","time"=>"2pm")); ?>).

```
$.post("test.php", { "func": "getNameAndTime" },
 function(data){
  console.log(data.name); // John
  console.log(data.time); // 2pm
 }, "json");
```

Example
Post a form using ajax and put results in a div

```
/* attach a submit handler to the form */
```

```
$("#searchForm").submit(function(event) {

 /* stop form from submitting normally */
 event.preventDefault();

 /* get some values from elements on the page: */
 var $form = $( this ),
   term = $form.find( 'input[name="s"]' ).val(),
   url = $form.attr( 'action' );

 /* Send the data using post and put the results in a div */
 $.post( url, { s: term },
  function( data ) {
    var content = $( data ).find( '#content' );
    $( "#result" ).empty().append( content );
  }
 );
});
```

slideUp([duration], [callback])

Hide the matched elements with a sliding motion.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideUp()` method animates the height of the matched elements. This causes lower parts of the page to slide up, appearing to conceal the items. Once the height reaches 0 (or, if set, to whatever the CSS min-height property is), the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially shown, we can hide it slowly:

```
$('#clickme').click(function() {
 $('#book').slideUp('slow', function() {
  // Animation complete.
 });
});
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example
Animates all divs to slide up, completing the animation within 400 milliseconds.

```
$(document.body).click(function () {
  if ($("div:first").is(":hidden")) {
   $("div").show("slow");
  } else {
   $("div").slideUp();
  }
 });
```

Example
Animates the parent paragraph to slide up, completing the animation within 200 milliseconds. Once the animation is done, it displays an alert.

```
$("button").click(function () {
  $(this).parent().slideUp("slow", function () {
   $("#msg").text($("button", this).text() + " has completed.");
  });
 });
```

next([selector])
Get the immediately following sibling of each element in the set of matched elements. If a selector is provided, it retrieves the next sibling only if it matches that selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.next()` method allows us to search through the immediately following sibling of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the immediately following sibling matches the selector, it remains in the newly constructed jQuery object; otherwise, it is excluded.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the element which comes just after it:

```
$('li.third-item').next().css('background-color', 'red');
```

The result of this call is a red background behind item 4. Since we do not supply a selector expression, this following element is unequivocally included as part of the object. If we had supplied one, the element would be tested for a match before it was included.

Example
Find the very next sibling of each disabled button and change its text "this button is disabled".

```
$("button[disabled]").next().text("this button is disabled");
```

Example
Find the very next sibling of each paragraph. Keep only the ones with a class "selected".

```
$("p").next(".selected").css("background", "yellow");
```

slideDown([duration], [callback])
Display the matched elements with a sliding motion.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideDown()` method animates the height of the matched elements. This causes lower parts of the page to slide down, making way for the revealed items.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially hidden, we can show it slowly:

```
$('#clickme').click(function() {
 $('#book').slideDown('slow', function() {
  // Animation complete.
 });
});
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example
Animates all divs to slide down and show themselves over 600 milliseconds.

```
$(document.body).click(function () {
if ($("div:first").is(":hidden")) {
$("div").slideDown("slow");
} else {
```

```
$("div").hide();
}
});
```

Example
Animates all inputs to slide down, completing the animation within 1000 milliseconds. Once the animation is done, the input look is changed especially if it is the middle input which gets the focus.

```
$("div").click(function () {
$(this).css({ borderStyle:"inset", cursor:"wait" });
$("input").slideDown(1000,function(){
$(this).css("border", "2px red inset")
.filter(".middle")
 .css("background", "yellow")
 .focus();
$("div").css("visibility", "hidden");
});
});
```

find(selector)
Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.find()` method allows us to search through the descendants of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.find()` and `.children()` methods are similar, except that the latter only travels a single level down the DOM tree.

The first signature for the `.find()`method accepts a selector expression of the same type that we can pass to the `$()` function. The elements will be filtered by testing whether they match this selector.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at item II, we can find list items within it:

```
$('li.item-ii').find('li').css('background-color', 'red');
```

The result of this call is a red background on items A, B, 1, 2, 3, and C. Even though item II matches the selector expression, it is not included in the results; only descendants are considered candidates for the match.

Unlike in the rest of the tree traversal methods, the selector expression is required in a call to `.find()`. If we need to retrieve all of the descendant

elements, we can pass in the universal selector `'*'` to accomplish this.

Selector context is implemented with the `.find()method;` therefore, `$('li.item-ii').find('li')` is equivalent to `$('li', 'li.item-ii')`.

As of jQuery 1.6, we can also filter the selection with a given jQuery collection or element. With the same nested list as above, if we start with:

```
var $allListElements = $('li');
```

And then pass this jQuery object to find:

```
$('li.item-ii').find( $allListElements );
```

This will return a jQuery collection which contains only the list elements that are descendants of item II.

Similarly, an element may also be passed to find:

```
var item1 = $('li.item-1')[0];
$('li.item-ii').find( item1 ).css('background-color', 'red');
```

The result of this call would be a red background on item 1.

Example
Starts with all paragraphs and searches for descendant span elements, same as $("p span")

```
$("p").find("span").css('color','red');
```

Example
A selection using a jQuery collection of all span tags. Only spans within p tags are changed to red while others are left blue.

```
var $spans = $('span');
 $("p").find( $spans ).css('color','red');
```

Example
Add spans around each word then add a hover and italicize words with the letter **t**.

```
var newText = $("p").text().split(" ").join("</span> <span>");
 newText = "<span>" + newText + "</span>";

 $("p").html( newText )
  .find('span')
  .hover(function() {
   $(this).addClass("hilite");
  },
   function() { $(this).removeClass("hilite");
  })
 .end()
  .find(":contains('t')")
  .css({"font-style":"italic", "font-weight":"bolder"});
```

jQuery.getScript(url, [success(script, textStatus, jqXHR)])
Load a JavaScript file from the server using a GET HTTP request, then execute it.

Arguments
url - A string containing the URL to which the request is sent.
success(script, textStatus, jqXHR) - A callback function that is executed if the request succeeds.

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
```

```
 url: url,
 dataType: "script",
 success: success
});
```

The script is executed in the global context, so it can refer to other variables and use jQuery functions. Included scripts can have some impact on the current page.

Success Callback

The callback is passed the returned JavaScript file. This is generally not useful as the script will already have run at this point.

```
$(".result").html("<p>Lorem ipsum dolor sit amet.</p>");
```

Scripts are included and run by referencing the file name:

```
$.getScript("ajax/test.js", function(data, textStatus, jqxhr) {
  console.log(data); //data returned
  console.log(textStatus); //success
  console.log(jqxhr.status); //200
  console.log('Load was performed.');
});
```

Handling Errors

As of jQuery 1.5, you may use .fail() to account for errors:

```
$.getScript("ajax/test.js")
.done(function(script, textStatus) {
 console.log( textStatus );
})
.fail(function(jqxhr, settings, exception) {
 $( "div.log" ).text( "Triggered ajaxError handler." );
});
```

Prior to jQuery 1.5, the global `.ajaxError()` callback event had to be used in order to handle `$.getScript()` errors:

```
$( "div.log" ).ajaxError(function(e, jqxhr, settings, exception) {
 if (settings.dataType=='script') {
  $(this).text( "Triggered ajaxError handler." );
 }
});
```

Caching Responses

Be default, `$.getScript()` sets the cache setting to `false`. This appends a timestamped query parameter to the request URL to ensure that the browser downloads the script each time it is requested. You can override this feature by setting the cache property globally using $.ajaxSetup():

```
$.ajaxSetup({
 cache: true
});
```

Alternatively, you could define a new method that uses the more flexible `$.ajax()` method.

Example
Define a $.cachedScript() method that allows fetching a cached script:

```
jQuery.cachedScript = function(url, options) {

 // allow user to set any option except for dataType, cache, and url
 options = $.extend(options || {}, {
  dataType: "script",
  cache: true,
  url: url
 });

 // Use $.ajax() since it is more flexible than $.getScript
 // Return the jqXHR object so we can chain callbacks
 return jQuery.ajax(options);
};

// Usage
$.cachedScript("ajax/test.js").done(function(script, textStatus) {
 console.log( textStatus );
});
```

Example
Load the official jQuery Color Animation plugin dynamically and bind some color animations to occur once the new functionality is loaded.

```
$.getScript("/scripts/jquery.color.js", function() {
 $("#go").click(function(){
  $(".block").animate( { backgroundColor: "pink" }, 1000)
   .delay(500)
   .animate( { backgroundColor: "blue" }, 1000);
 });
});
```

jQuery.getJSON(url, [data], [success(data, textStatus, jqXHR)])
Load JSON-encoded data from the server using a GET HTTP request.

Arguments
url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
 dataType: 'json',
 data: data,
 success: callback
});
```

Data that is sent to the server is appended to the URL as a query string. If the value of the `data` parameter is an object (map), it is converted to a string and url-encoded before it is appended to the URL.

Most implementations will specify a success handler:

```
$.getJSON('ajax/test.json', function(data) {
 var items = [];

 $.each(data, function(key, val) {
  items.push('<li id="' + key + '">' + val + '</li>');
 });
```

```
 $('<ul/>', {
  'class': 'my-new-list',
  html: items.join('')
 }).appendTo('body');
});
```

This example, of course, relies on the structure of the JSON file:

```
{
 "one": "Singular sensation",
 "two": "Beady little eyes",
 "three": "Little birds pitch by my doorstep"
}
```

Using this structure, the example loops through the requested data, builds an unordered list, and appends it to the body.

The `success` callback is passed the returned data, which is typically a JavaScript object or array as defined by the JSON structure and parsed using the `$.parseJSON()` method. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function receives a "jqXHR" object (in **jQuery 1.4**, it received the `XMLHttpRequest` object). However, since JSONP and cross-domain GET requests do not use XHR, in those cases the `jqXHR` and `textStatus` parameters passed to the success callback are undefined.

Important: As of jQuery 1.4, if the JSON file contains a syntax error, the request will usually fail silently. Avoid frequent hand-editing of JSON data for this reason. JSON is a data-interchange format with syntax rules that are stricter than those of JavaScript's object literal notation. For example, all strings represented in JSON, whether they are properties or values, must be enclosed in double-quotes. For details on the JSON format, see http://json.org/.

JSONP

If the URL includes the string "callback=?" (or similar, as defined by the server-side API), the request is treated as JSONP instead. See the discussion of the `jsonp` data type in `$.ajax()` for more details.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.getJSON()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.getJSON()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.getJSON("example.json", function() {
 alert("success");
})
.success(function() { alert("second success"); })
.error(function() { alert("error"); })
.complete(function() { alert("complete"); });

// perform other work here ...

// Set another completion function for the request above
jqxhr.complete(function(){ alert("second complete"); });
```

Example
Loads the four most recent cat pictures from the Flickr JSONP API.

```
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?",
 {
  tags: "cat",
  tagmode: "any",
  format: "json"
 },
 function(data) {
  $.each(data.items, function(i,item){
   $("<img/>").attr("src", item.media.m).appendTo("#images");
   if ( i == 3 ) return false;
  });
 });
```

Example
Load the JSON data from test.js and access a name from the returned JSON data.

```
$.getJSON("test.js", function(json) {
  alert("JSON Data: " + json.users[3].name);
 });
```

Example
Load the JSON data from test.js, passing along additional data, and access a name from the returned JSON data.

```
$.getJSON("test.js", { name: "John", time: "2pm" }, function(json) {
  alert("JSON Data: " + json.users[3].name);
  });
```

jQuery.get(url, [data], [success(data, textStatus, jqXHR)], [dataType])
Load data from the server using a HTTP GET request.

Arguments
url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.
dataType - The type of data expected from the server. Default: Intelligent Guess (xml, json, script, or html).

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
 data: data,
 success: success,
 dataType: dataType
});
```

The `success` callback function is passed the returned data, which will be an XML root element, text string, JavaScript file, or JSON object, depending on the MIME type of the response. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function is also passed a "jqXHR" object (in **jQuery 1.4**, it was passed the `XMLHttpRequest` object). However, since JSONP and cross-domain GET requests do not use XHR, in those cases the `(j)XHR` and `textStatus` parameters passed to the success callback are undefined.

Most implementations will specify a success handler:

```
$.get('ajax/test.html', function(data) {
 $('.result').html(data);
 alert('Load was performed.');
});
```

This example fetches the requested HTML snippet and inserts it on the page.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.get()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.get()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
 // and remember the jqxhr object for this request
 var jqxhr = $.get("example.php", function() {
  alert("success");
 })
 .success(function() { alert("second success"); })
 .error(function() { alert("error"); })
 .complete(function() { alert("complete"); });

 // perform other work here ...

 // Set another completion function for the request above
 jqxhr.complete(function(){ alert("second complete"); });
```

Example
Request the test.php page, but ignore the return results.

```
$.get("test.php");
```

Example
Request the test.php page and send some additional data along (while still ignoring the return results).

```
$.get("test.php", { name: "John", time: "2pm" } );
```

Example
pass arrays of data to the server (while still ignoring the return results).

```
$.get("test.php", { 'choices[]': ["Jon", "Susan"]} );
```

Example
Alert out the results from requesting test.php (HTML or XML, depending on what was returned).

```
$.get("test.php", function(data){
alert("Data Loaded: " + data);
});
```

Example
Alert out the results from requesting test.cgi with an additional payload of data (HTML or XML, depending on what was returned).

```
$.get("test.cgi", { name: "John", time: "2pm" },
  function(data){
   alert("Data Loaded: " + data);
  });
```

Example
Gets the test.php page contents, which has been returned in json format (<?php echo json_encode(array("name"=>"John","time"=>"2pm")); ?>), and adds it to the page.

```
$.get("test.php",
  function(data){
   $('body').append( "Name: " + data.name ) // John
        .append( "Time: " + data.time ); // 2pm
  }, "json");
```

load(url, data, [complete(responseText, textStatus, XMLHttpRequest)])

Load data from the server and place the returned HTML into the matched element.

Arguments

url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
complete(responseText, textStatus, XMLHttpRequest) - A callback function that is executed when the request completes.

Note: The event handling suite also has a method named `.load()`. jQuery determines which method to fire based on the set of arguments passed to it.

This method is the simplest way to fetch data from the server. It is roughly equivalent to `$.get(url, data, success)` except that it is a method rather than global function and it has an implicit callback function. When a successful response is detected (i.e. when `textStatus` is "success" or "notmodified"), `.load()` sets the HTML contents of the matched element to the returned data. This means that most uses of the method can be quite simple:

```
$('#result').load('ajax/test.html');
```

Callback Function

If a "complete" callback is provided, it is executed after post-processing and HTML insertion has been performed. The callback is fired once for each element in the jQuery collection, and `this` is set to each DOM element in turn.

```
$('#result').load('ajax/test.html', function() {
 alert('Load was performed.');
});
```

In the two examples above, if the current document does not contain an element with an ID of "result," the `.load()` method is not executed.

Request Method

The POST method is used if data is provided as an object; otherwise, GET is assumed.

Loading Page Fragments

The `.load()` method, unlike `$.get()`, allows us to specify a portion of the remote document to be inserted. This is achieved with a special syntax for the `url` parameter. If one or more space characters are included in the string, the portion of the string following the first space is assumed to be a jQuery selector that determines the content to be loaded.

We could modify the example above to use only part of the document that is fetched:

```
$('#result').load('ajax/test.html #container');
```

When this method executes, it retrieves the content of `ajax/test.html`, but then jQuery parses the returned document to find the element with an ID of `container`. This element, along with its contents, is inserted into the element with an ID of `result`, and the rest of the retrieved document is discarded.

jQuery uses the browser's `.innerHTML` property to parse the retrieved document and insert it into the current document. During this process, browsers often filter elements from the document such as `<html>`, `<title>`, or `<head>` elements. As a result, the elements retrieved by `.load()` may not be exactly the same as if the document were retrieved directly by the browser.

Script Execution

When calling `.load()` using a URL without a suffixed selector expression, the content is passed to `.html()` prior to scripts being removed. This executes the script blocks before they are discarded. If `.load()` is called with a selector expression appended to the URL, however, the scripts are stripped out prior to the DOM being updated, and thus are *not* executed. An example of both cases can be seen below:

Here, any JavaScript loaded into #a as a part of the document will successfully execute.

```
$('#a').load('article.html');
```

However, in the following case, script blocks in the document being loaded into #b are stripped out and not executed:

```
$('#b').load('article.html #target');
```

Example
Load the main page's footer navigation into an ordered list.

```
$("#new-nav").load("/ #jq-footerNavigation li");
```

Example
Display a notice if the Ajax request encounters an error.

```
$("#success").load("/not-here.php", function(response, status, xhr) {
 if (status == "error") {
  var msg = "Sorry but there was an error: ";
  $("#error").html(msg + xhr.status + " " + xhr.statusText);
 }
});
```

Example
Load the feeds.html file into the div with the ID of feeds.

```
$("#feeds").load("feeds.html");
```

Example
pass arrays of data to the server.

```
$("#objectID").load("test.php", { 'choices[]': ["Jon", "Susan"] } );
```

Example
Same as above, but will POST the additional parameters to the server and a callback that is executed when the server is finished responding.

```
$("#feeds").load("feeds.php", {limit: 25}, function(){
alert("The last 25 entries in the feed have been loaded");
});
```

jQuery.ajax(url, [settings])
Perform an asynchronous HTTP (Ajax) request.

Arguments
url - A string containing the URL to which the request is sent.
settings - A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup(). See jQuery.ajax(settings) below for a complete list of all settings.

The `$.ajax()` function underlies all Ajax requests sent by jQuery. It is often unnecessary to directly call this function, as several higher-level alternatives like `$.get()` and `.load()` are available and are easier to use. If less common options are required, though, `$.ajax()` can be used more flexibly.

At its simplest, the `$.ajax()` function can be called with no arguments:

```
$.ajax();
```

Note: Default settings can be set globally by using the `$.ajaxSetup()` function.

This example, using no options, loads the contents of the current page, but does nothing with the result. To use the result, we can implement one of the callback functions.

The jqXHR Object

The jQuery XMLHttpRequest (jqXHR) object returned by `$.ajax()` **as of jQuery 1.5** is a superset of the browser's native XMLHttpRequest object. For example, it contains `responseText` and `responseXML` properties, as well as a `getResponseHeader()` method. When the transport mechanism is something other than XMLHttpRequest (for example, a script tag for a JSONP request) the `jqXHR` object simulates native XHR functionality where possible.

As of jQuery 1.5.1, the `jqXHR` object also contains the `overrideMimeType()` method (it was available in jQuery 1.4.x, as well, but was temporarily removed in jQuery 1.5). The `.overrideMimeType()` method may be used in the `beforeSend()` callback function, for example, to modify the response content-type header:

```
$.ajax({
 url: "http://fiddle.jshell.net/favicon.png",
 beforeSend: function ( xhr ) {
  xhr.overrideMimeType("text/plain; charset=x-user-defined");
 }
}).done(function ( data ) {
 if( console && console.log ) {
  console.log("Sample of data:", data.slice(0, 100));
 }
});
```

The jqXHR objects returned by `$.ajax()` as of jQuery 1.5 implement the Promise interface, giving them all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, jqXHR also provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the `$.ajax()` request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback. This allows you to assign multiple callbacks on a single request, and even to assign callbacks after the request may have completed. (If the request is already complete, the callback is fired immediately.)

Deprecation Notice: The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callbacks will be deprecated in jQuery 1.8. To prepare your code for their eventual removal, use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.ajax( "example.php" )
  .done(function() { alert("success"); })
  .fail(function() { alert("error"); })
  .always(function() { alert("complete"); });

// perform other work here ...

// Set another completion function for the request above
jqxhr.always(function() { alert("second complete"); });
```

For backward compatibility with `XMLHttpRequest`, a `jqXHR` object will expose the following properties and methods:

- `readyState`
- `status`
- `statusText`
- `responseXML` and/or `responseText` when the underlying request responded with xml and/or text, respectively
- `setRequestHeader(name, value)` which departs from the standard by replacing the old value with the new one rather than concatenating the new value to the old one
- `getAllResponseHeaders()`
- `getResponseHeader()`

- `abort()`

No `onreadystatechange` mechanism is provided, however, since `success`, `error`, `complete` and `statusCode` cover all conceivable requirements.

Callback Function Queues

The `beforeSend`, `error`, `dataFilter`, `success` and `complete` options all accept callback functions that are invoked at the appropriate times.

As of jQuery 1.5, the `error` (`fail`), `success` (`done`), and `complete` (`always`, as of jQuery 1.6) callback hooks are first-in, first-out managed queues. This means you can assign more than one callback for each hook. See Deferred object methods, which are implemented internally for these `$.ajax()` callback hooks.

The `this` reference within all callbacks is the object in the `context` option passed to `$.ajax` in the settings; if `context` is not specified, `this` is a reference to the Ajax settings themselves.

Some types of Ajax requests, such as JSONP and cross-domain GET requests, do not use XHR; in those cases the `XMLHttpRequest` and `textStatus` parameters passed to the callback are `undefined`.

Here are the callback hooks provided by `$.ajax()`:

- `beforeSend` callback is invoked; it receives the `jqXHR` object and the `settings` map as parameters.
- `error` callbacks are invoked, in the order they are registered, if the request fails. They receive the `jqXHR`, a string indicating the error type, and an exception object if applicable. Some built-in errors will provide a string as the exception object: "abort", "timeout", "No Transport".
- `dataFilter` callback is invoked immediately upon successful receipt of response data. It receives the returned data and the value of `dataType`, and must return the (possibly altered) data to pass on to `success`.
- `success` callbacks are then invoked, in the order they are registered, if the request succeeds. They receive the returned data, a string containing the success code, and the `jqXHR` object.
- `complete` callbacks fire, in the order they are registered, when the request finishes, whether in failure or success. They receive the `jqXHR` object, as well as a string containing the success or error code.

For example, to make use of the returned HTML, we can implement a `success` handler:

```
$.ajax({
 url: 'ajax/test.html',
 success: function(data) {
  $('.result').html(data);
  alert('Load was performed.');
 }
});
```

Data Types

The `$.ajax()` function relies on the server to provide information about the retrieved data. If the server reports the return data as XML, the result can be traversed using normal XML methods or jQuery's selectors. If another type is detected, such as HTML in the example above, the data is treated as text.

Different data handling can be achieved by using the `dataType` option. Besides plain `xml`, the `dataType` can be `html`, `json`, `jsonp`, `script`, or `text`.

The `text` and `xml` types return the data with no processing. The data is simply passed on to the success handler, either through the `responseText` or `responseXML` property of the `jqXHR` object, respectively.

Note: We must ensure that the MIME type reported by the web server matches our choice of `dataType`. In particular, XML must be declared by the server as `text/xml` or `application/xml` for consistent results.

If `html` is specified, any embedded JavaScript inside the retrieved data is executed before the HTML is returned as a string. Similarly, `script` will execute the JavaScript that is pulled back from the server, then return nothing.

The `json` type parses the fetched data file as a JavaScript object and returns the constructed object as the result data. To do so, it uses `jQuery.parseJSON()` when the browser supports it; otherwise it uses a `Function`**constructor**. Malformed JSON data will throw a parse error (see json.org for more information). JSON data is convenient for communicating structured data in a way that is concise and easy for JavaScript to parse. If

the fetched data file exists on a remote server, specify the `jsonp` type instead.

The `jsonp` type appends a query string parameter of `callback=?` to the URL. The server should prepend the JSON data with the callback name to form a valid JSONP response. We can specify a parameter name other than `callback` with the `jsonp` option to `$.ajax()`.

Note: JSONP is an extension of the JSON format, requiring some server-side code to detect and handle the query string parameter. More information about it can be found in the original post detailing its use.

When data is retrieved from remote servers (which is only possible using the `script` or `jsonp` data types), the `error` callbacks and global events will never be fired.

Sending Data to the Server

By default, Ajax requests are sent using the GET HTTP method. If the POST method is required, the method can be specified by setting a value for the `type` option. This option affects how the contents of the `data` option are sent to the server. POST data will always be transmitted to the server using UTF-8 charset, per the W3C XMLHTTPRequest standard.

The `data` option can contain either a query string of the form `key1=value1&key2=value2`, or a map of the form `{key1: 'value1', key2: 'value2'}`. If the latter form is used, the data is converted into a query string using `jQuery.param()` before it is sent. This processing can be circumvented by setting `processData` to `false`. The processing might be undesirable if you wish to send an XML object to the server; in this case, change the `contentType` option from `application/x-www-form-urlencoded` to a more appropriate MIME type.

Advanced Options

The `global` option prevents handlers registered using `.ajaxSend()`, `.ajaxError()`, and similar methods from firing when this request would trigger them. This can be useful to, for example, suppress a loading indicator that was implemented with `.ajaxSend()` if the requests are frequent and brief. With cross-domain script and JSONP requests, the global option is automatically set to `false`. See the descriptions of these methods below for more details. See the descriptions of these methods below for more details.

If the server performs HTTP authentication before providing a response, the user name and password pair can be sent via the `username` and `password` options.

Ajax requests are time-limited, so errors can be caught and handled to provide a better user experience. Request timeouts are usually either left at their default or set as a global default using `$.ajaxSetup()` rather than being overridden for specific requests with the `timeout` option.

By default, requests are always issued, but the browser may serve results out of its cache. To disallow use of the cached results, set `cache` to `false`. To cause the request to report failure if the asset has not been modified since the last request, set `ifModified` to `true`.

The `scriptCharset` allows the character set to be explicitly specified for requests that use a `<script>` tag (that is, a type of `script` or `jsonp`). This is useful if the script and host page have differing character sets.

The first letter in Ajax stands for "asynchronous," meaning that the operation occurs in parallel and the order of completion is not guaranteed. The `async` option to `$.ajax()` defaults to `true`, indicating that code execution can continue after the request is made. Setting this option to `false` (and thus making the call no longer asynchronous) is strongly discouraged, as it can cause the browser to become unresponsive.

The `$.ajax()` function returns the `XMLHttpRequest` object that it creates. Normally jQuery handles the creation of this object internally, but a custom function for manufacturing one can be specified using the `xhr` option. The returned object can generally be discarded, but does provide a lower-level interface for observing and manipulating the request. In particular, calling `.abort()` on the object will halt the request before it completes.

At present, due to a bug in Firefox where `.getAllResponseHeaders()` returns the empty string although `.getResponseHeader('Content-Type')` returns a non-empty string, automatically decoding JSON CORS responses in Firefox with jQuery is not supported.

A workaround to this is possible by overriding `jQuery.ajaxSettings.xhr` as follows:

```
var _super = jQuery.ajaxSettings.xhr;
jQuery.ajaxSettings.xhr = function () {
  var xhr = _super(),
    getAllResponseHeaders = xhr.getAllResponseHeaders;
```

```
  xhr.getAllResponseHeaders = function () {
    if ( getAllResponseHeaders() ) {
      return getAllResponseHeaders();
    }
    var allHeaders = "";
    $( ["Cache-Control", "Content-Language", "Content-Type",
        "Expires", "Last-Modified", "Pragma"] ).each(function (i, header_name) {

      if ( xhr.getResponseHeader( header_name ) ) {
        allHeaders += header_name + ": " + xhr.getResponseHeader( header_name ) + "n";
      }
      return allHeaders;
    });
  };
  return xhr;
};
```

Extending Ajax

As of jQuery 1.5, jQuery's Ajax implementation includes prefilters, converters, and transports that allow you to extend Ajax with a great deal of flexibility. For more information about these advanced features, see the Extending Ajax page.

Example

Save some data to the server and notify the user once it's complete.

```
$.ajax({
 type: "POST",
 url: "some.php",
 data: { name: "John", location: "Boston" }
}).done(function( msg ) {
 alert( "Data Saved: " + msg );
});
```

Example

Retrieve the latest version of an HTML page.

```
$.ajax({
 url: "test.html",
 cache: false
}).done(function( html ) {
 $("#results").append(html);
});
```

Example

Send an xml document as data to the server. By setting the processData option to `false`, the automatic conversion of data to strings is prevented.

```
var xmlDocument = [create xml document];
var xmlRequest = $.ajax({
 url: "page.php",
 processData: false,
 data: xmlDocument
});

xmlRequest.done(handleResponse);
```

Example

Send an id as data to the server, save some data to the server, and notify the user once it's complete. If the request fails, alert the user.

```
var menuId = $("ul.nav").first().attr("id");
var request = $.ajax({
 url: "script.php",
```

```
 type: "POST",
 data: {id : menuId},
 dataType: "html"
});

request.done(function(msg) {
 $("#log").html( msg );
});

request.fail(function(jqXHR, textStatus) {
 alert( "Request failed: " + textStatus );
});
```

Example
Load and execute a JavaScript file.

```
$.ajax({
 type: "GET",
 url: "test.js",
 dataType: "script"
});
```

length
The number of elements in the jQuery object.

The number of elements currently matched. The .size() method will return the same value.

Example
Count the divs. Click to add more.

```
$(document.body).click(function () {
   $(document.body).append($("<div>"));
   var n = $("div").length;
   $("span").text("There are " + n + " divs." +
           "Click to add more.");
  }).trigger('click'); // trigger the click to start
```

children([selector])
Get the children of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.children()` method allows us to search through the children of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.children()` method differs from `.find()` in that `.children()` only travels a single level down the DOM tree while `.find()` can traverse down multiple levels to select descendant elements (grandchildren, etc.) as well. Note also that like most jQuery methods, `.children()` does not return text nodes; to get *all* children including text and comment nodes, use `.contents()`.

The `.children()` method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
```

```
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at the level-2 list, we can find its children:

```
$('ul.level-2').children().css('background-color', 'red');
```

The result of this call is a red background behind items A, B, and C. Since we do not supply a selector expression, all of the children are part of the returned jQuery object. If we had supplied one, only the matching items among these three would be included.

Example
Find all children of the clicked element.

```
$("#container").click(function (e) {
   $("*").removeClass("hilite");
   var $kids = $(e.target).children();
   var len = $kids.addClass("hilite").length;

   $("#results span:first").text(len);
   $("#results span:last").text(e.target.tagName);

   e.preventDefault();
   return false;
  });
```

Example
Find all children of each div.

```
$("div").children().css("border-bottom", "3px double red");
```

Example
Find all children with a class "selected" of each div.

```
$("div").children(".selected").css("color", "blue");
```

add(selector)
Add elements to the set of matched elements.

Arguments
selector - A string representing a selector expression to find additional elements to add to the set of matched elements.

Given a jQuery object that represents a set of DOM elements, the `.add()` method constructs a new jQuery object from the union of those elements and the ones passed into the method. The argument to `.add()` can be pretty much anything that `$()` accepts, including a jQuery selector expression, references to DOM elements, or an HTML snippet.

The updated set of elements can be used in a following (chained) method, or assigned to a variable for later use. For example:

```
$("p").add("div").addClass("widget");
var pdiv = $("p").add("div");
```

The following will *not* save the added elements, because the `.add()` method creates a new set and leaves the original set in pdiv unchanged:

```
var pdiv = $("p");
pdiv.add("div"); // WRONG, pdiv will not change
```

Consider a page with a simple list and a paragraph following it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
</ul>
<p>a paragraph</p>
```

We can select the list items and then the paragraph by using either a selector or a reference to the DOM element itself as the `.add()` method's argument:

```
$('li').add('p').css('background-color', 'red');
```

Or:

```
$('li').add(document.getElementsByTagName('p')[0])
 .css('background-color', 'red');
```

The result of this call is a red background behind all four elements.Using an HTML snippet as the `.add()` method's argument (as in the third version), we can create additional elements on the fly and add those elements to the matched set of elements. Let's say, for example, that we want to alter the background of the list items along with a newly created paragraph:

```
$('li').add('<p id="new">new paragraph</p>')
 .css('background-color', 'red');
```

Although the new paragraph has been created and its background color changed, it still does not appear on the page. To place it on the page, we could add one of the insertion methods to the chain.

As of jQuery 1.4 the results from .add() will always be returned in document order (rather than a simple concatenation).

Note: To reverse the `.add()` you can use .not(elements | selector) to remove elements from the jQuery results, or .end() to return to the selection before you added.

Example

Finds all divs and makes a border. Then adds all paragraphs to the jQuery object to set their backgrounds yellow.

```
$("div").css("border", "2px solid red")
    .add("p")
    .css("background", "yellow");
```

Example

Adds more elements, matched by the given expression, to the set of matched elements.

```
$("p").add("span").css("background", "yellow");
```

Example

Adds more elements, created on the fly, to the set of matched elements.

```
$("p").clone().add("<span>Again</span>").appendTo(document.body);
```

Example

Adds one or more Elements to the set of matched elements.

```
$("p").add(document.getElementById("a")).css("background", "yellow");
```

Example
Demonstrates how to add (or push) elements to an existing collection

```
var collection = $("p");
// capture the new collection
collection = collection.add(document.getElementById("a"));
collection.css("background", "yellow");
```

not(selector)
Remove elements from the set of matched elements.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.not()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; the elements that don't match the selector will be included in the result.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').not(':even').css('background-color', 'red');
```

The result of this call is a red background for items 2 and 4, as they do not match the selector (recall that :even and :odd use 0-based indexing).

Removing Specific Elements

The second version of the `.not()` method allows us to remove elements from the matched set, assuming we have found those elements previously by some other means. For example, suppose our list had an id applied to one of its items:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li id="notli">list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can fetch the third list item using the native JavaScript `getElementById()` function, then remove it from a jQuery object:

```
$('li').not(document.getElementById('notli'))
 .css('background-color', 'red');
```

This statement changes the color of items 1, 2, 4, and 5. We could have accomplished the same thing with a simpler jQuery expression, but this technique can be useful when, for example, other libraries provide references to plain DOM nodes.

As of jQuery 1.4, the `.not()` method can take a function as its argument in the same way that `.filter()` does. Elements for which the function

returns `true` are excluded from the filtered set; all other elements are included.

Example
Adds a border to divs that are not green or blue.

```
$("div").not(".green, #blueone")
      .css("border-color", "red");
```

Example
Removes the element with the ID "selected" from the set of all paragraphs.

```
$("p").not( $("#selected")[0] )
```

Example
Removes the element with the ID "selected" from the set of all paragraphs.

```
$("p").not("#selected")
```

Example
Removes all elements that match "div p.selected" from the total set of all paragraphs.

```
$("p").not($("div p.selected"))
```

toggle([duration], [callback])
Display or hide the matched elements.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

Note: The event handling suite also has a method named .toggle(). Which one is fired depends on the set of arguments passed.

With no parameters, the `.toggle()` method simply toggles the visibility of elements:

```
$('.target').toggle();
```

The matched elements will be revealed or hidden immediately, with no animation, by changing the CSS `display` property. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration is provided, `.toggle()` becomes an animation method. The `.toggle()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

We will cause `.toggle()` to be called when another element is clicked:

```
$('#clickme').click(function() {
 $('#book').toggle('slow', function() {
  // Animation complete.
 });
});
```

With the element initially shown, we can hide it slowly with the first click:

A second click will show the element once again:

The second version of the method accepts a Boolean parameter. If this parameter is `true`, then the matched elements are shown; if `false`, the elements are hidden. In essence, the statement:

```
$('#foo').toggle(showOrHide);
```

is equivalent to:

```
if ( showOrHide == true ) {
 $('#foo').show();
} else if ( showOrHide == false ) {
 $('#foo').hide();
}
```

Example
Toggles all paragraphs.

```
$("button").click(function () {
$("p").toggle();
});
```

Example
Animates all paragraphs to be shown if they are hidden and hidden if they are visible, completing the animation within 600 milliseconds.

```
$("button").click(function () {
$("p").toggle("slow");
});
```

Example
Shows all paragraphs, then hides them all, back and forth.

```
var flip = 0;
$("button").click(function () {
$("p").toggle( flip++ % 2 == 0 );
});
```

hide()
Hide the matched elements.

With no parameters, the `.hide()` method is the simplest way to hide an element:

```
$('.target').hide();
```

The matched elements will be hidden immediately, with no animation. This is roughly equivalent to calling `.css('display', 'none')`, except that the value of the `display` property is saved in jQuery's data cache so that `display` can later be restored to its initial value. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration is provided, `.hide()` becomes an animation method. The `.hide()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Note that `.hide()` is fired immediately and will override the animation queue if no duration or a duration of 0 is specified.

As of jQuery **1.4.3**, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
With the element initially shown, we can hide it slowly:
$('#clickme').click(function() {
 $('#book').hide('slow', function() {
  alert('Animation complete.');
 });
});
```

Example

Hides all paragraphs then the link on click.

```
$("p").hide();
  $("a").click(function ( event ) {
   event.preventDefault();
   $(this).hide();
  });
```

Example

Animates all shown paragraphs to hide slowly, completing the animation within 600 milliseconds.

```
$("button").click(function () {
   $("p").hide("slow");
  });
```

Example

Animates all spans (words in this case) to hide fastly, completing each animation within 200 milliseconds. Once each animation is done, it starts the next one.

```
$("#hidr").click(function () {
   $("span:last-child").hide("fast", function () {
    // use callee so don't have to name the function
    $(this).prev().hide("fast", arguments.callee);
   });
  });
  $("#showr").click(function () {
```

```
   $("span").show(2000);
  });
```

Example
Hides the divs when clicked over 2 seconds, then removes the div element when its hidden. Try clicking on more than one box at a time.

```
for (var i = 0; i < 5; i++) {
   $("<div>").appendTo(document.body);
  }
  $("div").click(function () {
   $(this).hide(2000, function () {
    $(this).remove();
   });
  });
```

width()
Get the current computed width for the first element in the set of matched elements.

The difference between `.css(width)` and `.width()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.width()` method is recommended when an element's width needs to be used in a mathematical calculation.

This method is also able to find the width of the window and document.

```
$(window).width();  // returns width of browser viewport
$(document).width(); // returns width of HTML document
```

Note that `.width()` will always return the content width, regardless of the value of the CSS `box-sizing` property.

Example
Show various widths. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showWidth(ele, w) {
 $("div").text("The width for the " + ele +
        " is " + w + "px.");
}
$("#getp").click(function () {
 showWidth("paragraph", $("p").width());
});
$("#getd").click(function () {
 showWidth("document", $(document).width());
});
$("#getw").click(function () {
 showWidth("window", $(window).width());
});
```

width(value)
Set the CSS width of each element in the set of matched elements.

Arguments
value - An integer representing the number of pixels, or an integer along with an optional unit of measure appended (as a string).

When calling `.width("value")`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, any valid CSS measurement may be used for the width (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS width property does not include padding, border, or margin, unless the `box-sizing` CSS property is used.

If no explicit unit is specified (like "em" or "%") then "px" is assumed.

Note that `.width("value")` sets the width of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the `outerWidth` of the box instead of the content width.

Example
Change the width of each div the first time it is clicked (and change its color).

```
(function() {
 var modWidth = 50;
 $("div").one('click', function () {
  $(this).width(modWidth).addClass("mod");
 modWidth -= 8;
 });
})();
```

height()
Get the current computed height for the first element in the set of matched elements.

The difference between `.css('height')` and `.height()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.height()` method is recommended when an element's height needs to be used in a mathematical calculation.

This method is also able to find the height of the window and document.

```
$(window).height();  // returns height of browser viewport
$(document).height(); // returns height of HTML document
```

Note that `.height()` will always return the content height, regardless of the value of the CSS `box-sizing` property.

Note: Although `style` and `script` tags will report a value for `.width()` or `height()` when absolutely positioned and given `display:block`, it is strongly discouraged to call those methods on these tags. In addition to being a bad practice, the results may also prove unreliable.

Example
Show various heights. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showHeight(ele, h) {
   $("div").text("The height for the " + ele +
          " is " + h + "px.");
  }
  $("#getp").click(function () {
   showHeight("paragraph", $("p").height());
  });
  $("#getd").click(function () {
   showHeight("document", $(document).height());
  });
  $("#getw").click(function () {
   showHeight("window", $(window).height());
  });
```

height(value)
Set the CSS height of every matched element.

Arguments
value - An integer representing the number of pixels, or an integer with an optional unit of measure appended (as a string).

When calling `.height(value)`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, a valid CSS measurement must be provided for the height (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS height property does not include padding, border, or margin.

If no explicit unit was specified (like 'em' or '%') then "px" is concatenated to the value.

Note that `.height(value)` sets the height of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the outerHeight of the box instead of the content height.

Example
To set the height of each div on click to 30px plus a color change.

```
$("div").one('click', function () {
   $(this).height(30)
       .css({cursor:"auto", backgroundColor:"green"});
  });
```

show()
Display the matched elements.

With no parameters, the `.show()` method is the simplest way to display an element:

```
$('.target').show();
```

The matched elements will be revealed immediately, with no animation. This is roughly equivalent to calling `.css('display', 'block')`, except that the `display` property is restored to whatever it was initially. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

Note: If using !important in your styles, such as`display: none !important`,it is necessary to override the style using `.css('display', 'block !important')` should you wish for `.show()` to function correctly.

When a duration is provided, `.show()` becomes an animation method. The `.show()` method animates the width, height, and opacity of the matched elements simultaneously.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
With the element initially hidden, we can show it slowly:
$('#clickme').click(function() {
 $('#book').show('slow', function() {
  // Animation complete.
 });
});
```

Example
Animates all hidden paragraphs to show slowly, completing the animation within 600 milliseconds.

```
$("button").click(function () {
```

```
  $("p").show("slow");
  });
```

Example
Show the first div, followed by each next adjacent sibling div in order, with a 200ms animation. Each animation starts when the previous sibling div's animation ends.

```
$("#showr").click(function () {
 $("div").first().show("fast", function showNext() {
  $(this).next("div").show("fast", showNext);
 });
});

$("#hidr").click(function () {
 $("div").hide(1000);
});
```

Example
Show all span and input elements with an animation. Change the text once the animation is done.

```
function doIt() {
 $("span,div").show("slow");
}
/* can pass in function name */
$("button").click(doIt);

$("form").submit(function () {
 if ($("input").val() == "yes") {
  $("p").show(4000, function () {
   $(this).text("Ok, DONE! (now showing)");
  });
 }
 $("span,div").hide("fast");
 /* to stop the submit */
 return false;
});
```

jQuery.trim(str)
Remove the whitespace from the beginning and end of a string.

Arguments
str - The string to trim.

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.

Example
Remove the two white spaces at the start and at the end of the string.

```
var str = "     lots of spaces before and after     ";
 $("#original").html("Original String: '" + str + "'");
 $("#trimmed").html("$.trim()'ed: '" + $.trim(str) + "'");
```

Example
Remove the two white spaces at the start and at the end of the string.

```
$.trim("  hello, how are you?  ");
```

jQuery.merge(first, second)
Merge the contents of two arrays together into the first array.

Arguments
first - The first array to merge, the elements of second added.
second - The second array to merge into the first, unaltered.

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 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:

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

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

Prior to jQuery 1.4, the arguments should be true Javascript Array objects; use `$.makeArray` if they are not.

Example
Merges two arrays, altering the first argument.

```
$.merge( [0,1,2], [2,3,4] )
```

Example
Merges two arrays, altering the first argument.

```
$.merge( [3,2,1], [4,3,2] )
```

Example
Merges two arrays, but uses a copy, so the original isn't altered.

```
var first = ['a','b','c'];
var second = ['d','e','f'];
$.merge( $.merge([],first), second);
```

jQuery.map(array, callback(elementOfArray, indexInArray))
Translate all items in an array or object to new array of items.

Arguments
array - The Array to translate.
callback(elementOfArray, indexInArray) - The function to process each item against. The first argument to the function is the array item, the second argument is the index in array The function can return any value. Within the function, `this` refers to the global (window) object.

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 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 - must be converted to actual arrays before being passed to `$.map()`. The jQuery library provides $.makeArray() for such conversions.

```
// The following object masquerades as an array.
var fakeArray = {"length": 1, 0: "Addy", 1: "Subtracty"};

// Therefore, convert it to a real array
var realArray = $.makeArray( fakeArray )

// Now it can be used reliably with $.map()
$.map( realArray, function(val, i) {
 // do something
});
```

The translation function that is provided to this method is called for each top-level element in the array or object and is passed two arguments: The element's value and its index or key within the array or object.

The function can return:

- the translated value, which will be mapped to the resulting array
- `null`, to remove the item
- an array of values, which will be flattened into the full array

Example

A couple examples of using .map()

```
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(", "));
```

Example

Map the original array to a new one and add 4 to each value.

```
$.map( [0,1,2], function(n){
  return n + 4;
 });
```

Example

Maps the original array to a new one and adds 1 to each value if it is bigger then zero, otherwise it's removed.

```
$.map( [0,1,2], function(n){
  return n > 0 ? n + 1 : null;
 });
```

Example

Map the original array to a new one; each element is added with its original value and the value plus one.

```
$.map( [0,1,2], function(n){
  return [ n, n + 1 ];
 });
```

Example

Map the original object to a new array and double each value.

```
var dimensions = { width: 10, height: 15, length: 20 };
dimensions = $.map( dimensions, function( value, index ) {
 return value * 2;
});
```

Example

Map an object's keys to an array.

```
var dimensions = { width: 10, height: 15, length: 20 },
  keys = $.map( dimensions, function( value, index ) {
   return index;
  });
```

Example

Maps the original array to a new one; each element is squared.

```
$.map( [0,1,2,3], function (a) {
 return a * a;
```

```
});
```

Example
Remove items by returning `null` from the function. This removes any numbers less than 50, and the rest are decreased by 45.

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

Example
Augmenting the resulting array by returning an array inside the function.

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

jQuery.grep(array, function(elementOfArray, indexInArray), [invert])
Finds the elements of an array which satisfy a filter function. The original array is not affected.

Arguments
array - The array to search through.
function(elementOfArray, indexInArray) - The function to process each item against. The first argument to the function is the item, and the second argument is the index. The function should return a Boolean value. `this` will be the global window object.
invert - If "invert" is false, or not provided, then the function returns an array consisting of all elements for which "callback" returns true. If "invert" is true, then the function returns an array consisting of all elements for which "callback" returns false.

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.

Example
Filters the original array of numbers leaving that are not 5 and have an index greater than 4. Then it removes all 9s.

```
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(", "));
```

Example
Filter an array of numbers to include only numbers bigger then zero.

```
$.grep( [0,1,2], function(n,i){
  return n > 0;
 });
```

Example
Filter an array of numbers to include numbers that are not bigger than zero.

```
$.grep( [0,1,2], function(n,i){
  return n > 0;
},true);
```

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

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

Arguments

target - An object that will receive the new properties if additional objects are passed in or that will extend the jQuery namespace if it is the sole argument.
object1 - An object containing additional properties to merge in.
objectN - Additional objects containing properties to merge in.

When we supply two or more objects to `$.extend()`, 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. 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, we want to preserve both of the original objects, we can do so by passing an empty object as the target:

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

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 example below by examining the value of banana. However, by passing `true` for the first function argument, objects will be recursively merged. (Passing `false` for the first argument is not supported.)

Undefined properties are not copied. However, properties inherited from the object's prototype *will* be copied over. Properties that are an object constructed via `new MyCustomObject(args)`, or built-in JavaScript types such as Date or RegExp, are not re-constructed and will appear as plain Objects in the resulting object or array.

On a `deep` extend, Object and Array are extended, but object wrappers on primitive types such as String, Boolean, and Number are not.

For needs that fall outside of this behavior, write a custom extend method instead.

Example

Merge two objects, modifying the first.

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

/* merge object2 into object1 */
$.extend(object1, object2);

var printObj = typeof JSON != "undefined" ? JSON.stringify : function(obj) {
 var arr = [];
 $.each(obj, function(key, val) {
  var next = key + ": ";
  next += $.isPlainObject(val) ? printObj(val) : val;
  arr.push( next );
 });
 return "{ " + arr.join(", ") + " }";
};

$("#log").append( printObj(object1) );
```

Example

Merge two objects recursively, modifying the first.

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

/* merge object2 into object1, recursively */
$.extend(true, object1, object2);

var printObj = typeof JSON != "undefined" ? JSON.stringify : function(obj) {
 var arr = [];
 $.each(obj, function(key, val) {
  var next = key + ": ";
  next += $.isPlainObject(val) ? printObj(val) : val;
  arr.push( next );
 });
 return "{ " + arr.join(", ") + " }";
};

$("#log").append( printObj(object1) );
```

Example

Merge defaults and options, without modifying the defaults. This is a common plugin development pattern.

```
var defaults = { validate: false, limit: 5, name: "foo" };
var options = { validate: true, name: "bar" };

/* merge defaults and options, without modifying defaults */
var settings = $.extend({}, defaults, options);

var printObj = typeof JSON != "undefined" ? JSON.stringify : function(obj) {
 var arr = [];
 $.each(obj, function(key, val) {
  var next = key + ": ";
  next += $.isPlainObject(val) ? printObj(val) : val;
  arr.push( next );
 });
 return "{ " + arr.join(", ") + " }";
};


$("#log").append( "<div><b>defaults -- </b>" + printObj(defaults) + "</div>" );
$("#log").append( "<div><b>options -- </b>" + printObj(options) + "</div>" );
$("#log").append( "<div><b>settings -- </b>" + printObj(settings) + "</div>" );
```

jQuery.each(collection, callback(indexInArray, valueOfElement))

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.

Arguments

collection - The object or array to iterate over.
callback(indexInArray, valueOfElement) - The function that will be executed on every object.

The `$.each()` function is not the same as $(selector).each(), which is used to iterate, exclusively, over a jQuery object. The `$.each()` function can be used to iterate over any collection, whether it is 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. (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: 521: 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(key, value) {
 alert(key + ': ' + value);
});
```

Once again, this produces two messages:

flammable: inflammableduh: no duh

We can break the `$.each()` loop at a particular iteration by making the callback function return `false`. Returning *non-false* is the same as a `continue` statement in a for loop; it will skip immediately to the next iteration.

Example
Iterates through the array displaying each number as both a word and numeral

```
var arr = [ "one", "two", "three", "four", "five" ];
  var obj = { one:1, two:2, three:3, four:4, five:5 };

  jQuery.each(arr, function() {
   $("#" + this).text("Mine is " + this + ".");
    return (this != "three"); // will stop running after "three"
  });

  jQuery.each(obj, function(i, val) {
   $("#" + i).append(document.createTextNode(" - " + val));
  });
```

Example
Iterates over items in an array, accessing both the current item and its index.

```
$.each( ['a','b','c'], function(i, l){
  alert( "Index #" + i + ": " + l );
 });
```

Example
Iterates over the properties in an object, accessing both the current item and its key.

```
$.each( { name: "John", lang: "JS" }, function(k, v){
  alert( "Key: " + k + ", Value: " + v );
 });
```

jQuery.boxModel
Deprecated in jQuery 1.3 (see jQuery.support). States if the current page, in the user's browser, is being rendered using the W3C CSS Box Model.

Example
Returns the box model for the iframe.

```
$("p").html("The box model for this iframe is: <span>" +
        jQuery.boxModel + "</span>");
```

Example
Returns false if the page is in Quirks Mode in Internet Explorer

```
$.boxModel
```

css(propertyName)
Get the value of a style property for the first element in the set of matched elements.

Arguments
propertyName - A CSS property.

The `.css()` method is a convenient way to get a style property from the first matched element, especially in light of the different ways browsers access most of those properties (the `getComputedStyle()` method in standards-based browsers versus the `currentStyle` and `runtimeStyle` properties in Internet Explorer) and the different terms browsers use for certain properties. For example, Internet Explorer's DOM implementation refers to the `float` property as `styleFloat`, while W3C standards-compliant browsers refer to it as `cssFloat`. The `.css()` method accounts for such differences, producing the same result no matter which term we use. For example, an element that is floated left will return the string `left` for each of the following three lines:

- `$('div.left').css('float');`
- `$('div.left').css('cssFloat');`
- `$('div.left').css('styleFloat');`

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css('background-color')` and `.css('backgroundColor')`. Different browsers may return CSS color values that are logically but not textually equal, e.g., #FFF, #ffffff, and rgb(255,255,255).

Shorthand CSS properties (e.g. margin, background, border) are not supported. For example, if you want to retrieve the rendered margin, use: `$(elem).css('marginTop')` and `$(elem).css('marginRight')`, and so on.

Example
To access the background color of a clicked div.

```
$("div").click(function () {
 var color = $(this).css("background-color");
 $("#result").html("That div is <span style='color:" +
           color + ";'>" + color + "</span>.");
});
```

css(propertyName, value)
Set one or more CSS properties for the set of matched elements.

Arguments
propertyName - A CSS property name.
value - A value to set for the property.

As with the `.prop()` method, the `.css()` method makes setting properties of elements quick and easy. This method can take either a property name and value as separate parameters, or a single map of key-value pairs (JavaScript object notation).

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css({'background-color': '#ffe', 'border-left': '5px solid #ccc'})` and `.css({backgroundColor: '#ffe', borderLeft: '5px solid #ccc'})`. Notice that with the DOM notation, quotation marks around the property names are optional, but with CSS notation they're required due to the hyphen in the name.

When using `.css()` as a setter, jQuery modifies the element's `style` property. For example, `$('#mydiv').css('color', 'green')` is equivalent to `document.getElementById('mydiv').style.color = 'green'`. Setting the value of a style property to an empty string - e.g. `$('#mydiv').css('color', '')` - removes that property from an element if it has already been directly applied, whether in the HTML style attribute, through jQuery's `.css()` method, or through direct DOM manipulation of the `style` property. It does not, however, remove a style that has been applied with a CSS rule in a stylesheet or `<style>` element.

As of jQuery 1.6, `.css()` accepts relative values similar to `.animate()`. Relative values are a string starting with `+=` or `-=` to increment or decrement the current value. For example, if an element's padding-left was 10px, `.css( "padding-left", "+=15" )` would result in a total padding-left of 25px.

As of jQuery 1.4, `.css()` allows us to pass a function as the property value:

```
$('div.example').css('width', function(index) {
 return index * 50;
});
```

This example sets the widths of the matched elements to incrementally larger values.

Note: If nothing is returned in the setter function (ie. `function(index, style){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example
To change the color of any paragraph to red on mouseover event.

```
$("p").mouseover(function () {
  $(this).css("color","red");
 });
```

Example
Increase the width of #box by 200 pixels

```
$("#box").one( "click", function () {
  $( this ).css( "width","+=200" );
 });
```

Example
To highlight a clicked word in the paragraph.

```
var words = $("p:first").text().split(" ");
 var text = words.join("</span> <span>");
 $("p:first").html("<span>" + text + "</span>");
 $("span").click(function () {
  $(this).css("background-color","yellow");
 });
```

Example
To set the color of all paragraphs to red and background to blue:

```
$("p").hover(function () {
  $(this).css({'background-color' : 'yellow', 'font-weight' : 'bolder'});
 }, function () {
  var cssObj = {
   'background-color' : '#ddd',
   'font-weight' : '',
   'color' : 'rgb(0,40,244)'
  }
  $(this).css(cssObj);
 });
```

Example
Increase the size of a div when you click it:

```
$("div").click(function() {
  $(this).css({
   width: function(index, value) {
    return parseFloat(value) * 1.2;
   },
   height: function(index, value) {
    return parseFloat(value) * 1.2;
```

```
   }

  });
 });
```

clone([withDataAndEvents])

Create a deep copy of the set of matched elements.

Arguments

withDataAndEvents - A Boolean indicating whether event handlers should be copied along with the elements. As of jQuery 1.4, element data will be copied as well.

The `.clone()` method performs a *deep* copy of the set of matched elements, meaning that it copies the matched elements as well as all of their descendant elements and text nodes. When used in conjunction with one of the insertion methods, `.clone()` is a convenient way to duplicate elements on a page. Consider the following HTML:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">Goodbye</div>
</div>
```

As shown in the discussion for `.append()`, normally when an element is inserted somewhere in the DOM, it is moved from its old location. So, given the code:

```
$('.hello').appendTo('.goodbye');
```

The resulting DOM structure would be:

```
<div class="container">
 <div class="goodbye">
  Goodbye
  <div class="hello">Hello</div>
 </div>
</div>
```

To prevent this and instead create a copy of the element, you could write the following:

```
$('.hello').clone().appendTo('.goodbye');
```

This would produce:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">
  Goodbye
  <div class="hello">Hello</div>
 </div>
</div>
```

Note: When using the `.clone()` method, you can modify the cloned elements or their contents before (re-)inserting them into the document.

Normally, any event handlers bound to the original element are *not* copied to the clone. The optional `withDataAndEvents` parameter allows us to change this behavior, and to instead make copies of all of the event handlers as well, bound to the new copy of the element. As of jQuery 1.4, all element data (attached by the `.data()` method) is also copied to the new copy.

However, objects and arrays within element data are not copied and will continue to be shared between the cloned element and the original element. To deep copy all data, copy each one manually:

```
var $elem = $('#elem').data( "arr": [ 1 ] ), // Original element with attached data
  $clone = $elem.clone( true )
  .data( "arr", $.extend( [], $elem.data("arr") ) ); // Deep copy to prevent data sharing
```

As of jQuery 1.5, `withDataAndEvents` can be optionally enhanced with `deepWithDataAndEvents` to copy the events and data for all children of the cloned element.

Note: Using `.clone()` has the side-effect of producing elements with duplicate `id` attributes, which are supposed to be unique. Where possible, it is recommended to avoid cloning elements with this attribute or using `class` attributes as identifiers instead.

Example
Clones all b elements (and selects the clones) and prepends them to all paragraphs.

```
$("b").clone().prependTo("p");
```

Example
When using `.clone()` to clone a collection of elements that are not attached to the DOM, their order when inserted into the DOM is not guaranteed. However, it may be possible to preserve sort order with a workaround, as demonstrated:

```
// sort order is not guaranteed here and may vary with browser
$('#copy').append($('#orig .elem')
     .clone()
     .children('a')
     .prepend('foo - ')
     .parent()
     .clone());

// correct way to approach where order is maintained
$('#copy-correct')
     .append($('#orig .elem')
     .clone()
     .children('a')
     .prepend('bar - ')
     .end());
```

remove([selector])
Remove the set of matched elements from the DOM.

Arguments
selector - A selector expression that filters the set of matched elements to be removed.

Similar to `.empty()`, the `.remove()` method takes elements out of the DOM. Use `.remove()` when you want to remove the element itself, as well as everything inside it. In addition to the elements themselves, all bound events and jQuery data associated with the elements are removed. To remove the elements without removing data and events, use `.detach()` instead.

Consider the following HTML:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">Goodbye</div>
</div>
```

We can target any element for removal:

```
$('.hello').remove();
```

This will result in a DOM structure with the `<div>` element deleted:

```
<div class="container">
 <div class="goodbye">Goodbye</div>
</div>
```

If we had any number of nested elements inside `<div class="hello">`, they would be removed, too. Other jQuery constructs such as data or event handlers are erased as well.

We can also include a selector as an optional parameter. For example, we could rewrite the previous DOM removal code as follows:

```
$('div').remove('.hello');
```

This would result in the same DOM structure:

```
<div class="container">
 <div class="goodbye">Goodbye</div>
</div>
```

Example
Removes all paragraphs from the DOM

```
$("button").click(function () {
   $("p").remove();
  });
```

Example
Removes all paragraphs that contain "Hello" from the DOM. Analogous to doing `$("p").filter(":contains('Hello')").remove()`.

```
$("button").click(function () {
   $("p").remove(":contains('Hello')");
  });
```

empty()
Remove all child nodes of the set of matched elements from the DOM.

This method removes not only child (and other descendant) elements, but also any text within the set of matched elements. This is because, according to the DOM specification, any string of text within an element is considered a child node of that element. Consider the following HTML:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">Goodbye</div>
</div>
```

We can target any element for removal:

```
$('.hello').empty();
```

This will result in a DOM structure with the `Hello` text deleted:

```
<div class="container">
 <div class="hello"></div>
 <div class="goodbye">Goodbye</div>
</div>
```

If we had any number of nested elements inside `<div class="hello">`, they would be removed, too.

To avoid memory leaks, jQuery removes other constructs such as data and event handlers from the child elements before removing the elements themselves.

If you want to remove elements without destroying their data or event handlers (so they can be re-added later), use .detach() instead.

Example
Removes all child nodes (including text nodes) from all paragraphs

```
$("button").click(function () {
  $("p").empty();
 });
```

wrap(wrappingElement)
Wrap an HTML structure around each element in the set of matched elements.

Arguments

wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the matched elements.

The `.wrap()` function can take any string or object that could be passed to the `$()` factory function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. A copy of this structure will be wrapped around each of the elements in the set of matched elements. This method returns the original set of elements for chaining purposes.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using `.wrap()`, we can insert an HTML structure around the inner `<div>` elements like so:

```
$('.inner').wrap('<div class="new" />');
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around each matched element:

```
<div class="container">
 <div class="new">
  <div class="inner">Hello</div>
 </div>
 <div class="new">
  <div class="inner">Goodbye</div>
 </div>
</div>
```

The second version of this method allows us to instead specify a callback function. This callback function will be called once for every matched element; it should return a DOM element, jQuery object, or HTML snippet in which to wrap the corresponding element. For example:

```
$('.inner').wrap(function() {
 return '<div class="' + $(this).text() + '" />';
});
```

This will cause each `<div>` to have a class corresponding to the text it wraps:

```
<div class="container">
 <div class="Hello">
  <div class="inner">Hello</div>
 </div>
 <div class="Goodbye">
  <div class="inner">Goodbye</div>
 </div>
</div>
```

Example

Wrap a new div around all of the paragraphs.

```
$("p").wrap("<div></div>");
```

Example

Wraps a newly created tree of objects around the spans. Notice anything in between the spans gets left out like the (red text) in this example. Even the white space between spans is left out. Click View Source to see the original html.

```
$("span").wrap("<div><div><p><em><b></b></em></p></div></div>");
```

Example

Wrap a new div around all of the paragraphs.

```
$("p").wrap(document.createElement("div"));
```

Example
Wrap a jQuery object double depth div around all of the paragraphs. Notice it doesn't move the object but just clones it to wrap around its target.

```
$("p").wrap($(".doublediv"));
```

insertBefore(target)
Insert every element in the set of matched elements before the target.

Arguments
target - A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted before the element(s) specified by this parameter.

The `.before()` and `.insertBefore()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.before()`, the selector expression preceding the method is the container before which the content is inserted. With `.insertBefore()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted before the target container.

Consider the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

We can create content and insert it before several elements at once:

```
$('<p>Test</p>').insertBefore('.inner');
```

Each inner `<div>` element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <p>Test</p>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
</div>
```

We can also select an element on the page and insert it before another:

```
$('h2').insertBefore($('.container'));
```

If an element selected this way is inserted elsewhere, it will be moved before the target (not cloned):

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

Example
Inserts all paragraphs before an element with id of "foo". Same as $("#foo").before("p")

```
$("p").insertBefore("#foo"); // check before() examples
```

before(content, [content])
Insert content, specified by the parameter, before each element in the set of matched elements.

Arguments

content - HTML string, DOM element, or jQuery object to insert before each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert before each element in the set of matched elements.

The `.before()` and `.insertBefore()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.before()`, the selector expression preceding the method is the container before which the content is inserted. With `.insertBefore()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted before the target container.

Consider the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it before several elements at once:

```
$('.inner').before('<p>Test</p>');
```

Each inner `<div>` element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <p>Test</p>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
</div>
```

You can also select an element on the page and insert it before another:

```
$('.container').before($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved before the target (not cloned):

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

In jQuery 1.4, `.before()` and `.after()` will also work on disconnected DOM nodes:

```
$("<div/>").before("<p></p>");
```

The result is a jQuery set that contains a paragraph and a div (in that order).

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.after()`, `.before()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` before the first paragraph:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');
```

```
$('p').first().before($newdiv1, [newdiv2, existingdiv1]);
```

Since `.before()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('p').first().before($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Inserts some HTML before all paragraphs.

```
$("p").before("<b>Hello</b>");
```

Example
Inserts a DOM element before all paragraphs.

```
$("p").before( document.createTextNode("Hello") );
```

Example
Inserts a jQuery object (similar to an Array of DOM Elements) before all paragraphs.

```
$("p").before( $("b") );
```

insertAfter(target)
Insert every element in the set of matched elements after the target.

Arguments
target - A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted after the element(s) specified by this parameter.

The `.after()` and `.insertAfter()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.after()`, the selector expression preceding the method is the container after which the content is inserted. With `.insertAfter()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted after the target container.

Consider the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

We can create content and insert it after several elements at once:

```
$('<p>Test</p>').insertAfter('.inner');
```

Each inner `<div>` element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
 <p>Test</p>
</div>
```

We can also select an element on the page and insert it after another:

```
$('h2').insertAfter($('.container'));
```

If an element selected this way is inserted elsewhere, it will be moved after the target (not cloned):

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
<h2>Greetings</h2>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

Example
Inserts all paragraphs after an element with id of "foo". Same as $("#foo").after("p")

```
$("p").insertAfter("#foo"); // check after() examples
```

after(content, [content])
Insert content, specified by the parameter, after each element in the set of matched elements.

Arguments
content - HTML string, DOM element, or jQuery object to insert after each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert after each element in the set of matched elements.

The `.after()` and `.insertAfter()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.after()`, the selector expression preceding the method is the container after which the content is inserted. With `.insertAfter()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted after the target container.

Using the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Content can be created and then inserted after several elements at once:

```
$('.inner').after('<p>Test</p>');
```

Each inner `<div>` element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
 <p>Test</p>
</div>
```

An element in the DOM can also be selected and inserted after another element:

```
$('.container').after($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved rather than cloned:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
<h2>Greetings</h2>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Inserting Disconnected DOM nodes

As of jQuery 1.4, `.before()` and `.after()` will also work on disconnected DOM nodes. For example, given the following code:

```
$('<div/>').after('<p></p>');
```

The result is a jQuery set containing a div and a paragraph, in that order. That set can be further manipulated, even before it is inserted in the document.

```
$('<div/>').after('<p></p>').addClass('foo')
 .filter('p').attr('id', 'bar').html('hello')
.end()
.appendTo('body');
```

This results in the following elements inserted just before the closing `</body>` tag:

```
<div class="foo"></div>
<p class="foo" id="bar">hello</p>
```

Passing a Function

As of jQuery 1.4, `.after()` supports passing a function that returns the elements to insert.

```
$('p').after(function() {
 return '<div>' + this.className + '</div>';
});
```

This example inserts a `<div>` after each paragraph, with each new `<div>` containing the class name(s) of its preceding paragraph.

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.before()`, `.after()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` after the first paragraph:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('p').first().after($newdiv1, [newdiv2, existingdiv1]);
```

Since `.after()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('p').first().after($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on the elements that are collected in the code.

Example

Inserts some HTML after all paragraphs.

```
$("p").after("<b>Hello</b>");
```

Example

Inserts a DOM element after all paragraphs.

```
$("p").after( document.createTextNode("Hello") );
```

Example

Inserts a jQuery object (similar to an Array of DOM Elements) after all paragraphs.

```
$("p").after( $("b") );
```

prependTo(target)

Insert every element in the set of matched elements to the beginning of the target.

Arguments

target - A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted at the beginning of the element(s) specified by this parameter.

The `.prepend()` and `.prependTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.prepend()`, the selector expression preceding the method is the container into which the content is inserted. With `.prependTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

We can create content and insert it into several elements at once:

```
$('<p>Test</p>').prependTo('.inner');
```

Each inner `<div>` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
  <p>Test</p>
  Hello
 </div>
 <div class="inner">
  <p>Test</p>
  Goodbye
 </div>
</div>
```

We can also select an element on the page and insert it into another:

```
$('h2').prependTo($('.container'));
```

If an element selected this way is inserted elsewhere, it will be moved into the target (not cloned):

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Example

Prepends all spans to the element with the ID "foo"

```
$("span").prependTo("#foo"); // check prepend() examples
```

prepend(content, [content])

Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

Arguments

content - DOM element, array of elements, HTML string, or jQuery object to insert at the beginning of each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert at the beginning of each element in the set of matched elements.

The `.prepend()` method inserts the specified content as the first child of each element in the jQuery collection (To insert it as the *last* child, use .append()).

The `.prepend()` and `.prependTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.prepend()`, the selector expression preceding the method is the container into which the content is inserted. With `.prependTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it into several elements at once:

```
$('.inner').prepend('<p>Test</p>');
```

Each `<div class="inner">` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
  <p>Test</p>
  Hello
 </div>
 <div class="inner">
  <p>Test</p>
  Goodbye
 </div>
</div>
```

You can also select an element on the page and insert it into another:

```
$('.container').prepend($('h2'));
```

If *a single element* selected this way is inserted elsewhere, it will be moved into the target (*not cloned*):

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

Important: If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Additional Arguments

Similar to other content-adding methods such as `.append()` and `.before()`, `.prepend()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` as the first three child nodes of the body:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('body').prepend($newdiv1, [newdiv2, existingdiv1]);
```

Since `.prepend()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('body').prepend($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Prepends some HTML to all paragraphs.

```
$("p").prepend("<b>Hello </b>");
```

Example
Prepends a DOM Element to all paragraphs.

```
$("p").prepend(document.createTextNode("Hello "));
```

Example
Prepends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
$("p").prepend( $("b") );
```

appendTo(target)
Insert every element in the set of matched elements to the end of the target.

Arguments
target - A selector, element, HTML string, or jQuery object; the matched set of elements will be inserted at the end of the element(s) specified by this parameter.

The `.append()` and `.appendTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.append()`, the selector expression preceding the method is the container into which the content is inserted. With `.appendTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

We can create content and insert it into several elements at once:

```
$('<p>Test</p>').appendTo('.inner');
```

Each inner `<div>` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
  Hello
  <p>Test</p>
 </div>
 <div class="inner">
  Goodbye
```

```
  <p>Test</p>
 </div>
</div>
```

We can also select an element on the page and insert it into another:

```
$('h2').appendTo($('.container'));
```

If an element selected this way is inserted elsewhere, it will be moved into the target (not cloned):

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
 <h2>Greetings</h2>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first, and that new set (the original element plus clones) is returned.

Example
Appends all spans to the element with the ID "foo"

```
$("span").appendTo("#foo"); // check append() examples
```

append(content, [content])
Insert content, specified by the parameter, to the end of each element in the set of matched elements.

Arguments
content - DOM element, HTML string, or jQuery object to insert at the end of each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert at the end of each element in the set of matched elements.

The `.append()` method inserts the specified content as the last child of each element in the jQuery collection (To insert it as the *first* child, use .prepend()).

The `.append()` and `.appendTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.append()`, the selector expression preceding the method is the container into which the content is inserted. With `.appendTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it into several elements at once:

```
$('.inner').append('<p>Test</p>');
```

Each inner `<div>` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
```

```
  Hello
  <p>Test</p>
 </div>
 <div class="inner">
  Goodbye
  <p>Test</p>
 </div>
</div>
```

You can also select an element on the page and insert it into another:

```
$('.container').append($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved into the target (not cloned):

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
 <h2>Greetings</h2>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.before()`, `.append()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` as the last three child nodes of the body:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('body').append($newdiv1, [newdiv2, existingdiv1]);
```

Since `.append()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('body').append($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Appends some HTML to all paragraphs.

```
$("p").append("<strong>Hello</strong>");
```

Example
Appends an Element to all paragraphs.

```
$("p").append(document.createTextNode("Hello"));
```

Example
Appends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
$("p").append( $("strong") );
```

val()
Get the current value of the first element in the set of matched elements.

The .val() method is primarily used to get the values of form elements such as input, select and textarea. In the case of <select multiple="multiple"> elements, the .val() method returns an array containing each selected option; if no option is selected, it returns null.

For selects and checkboxes, you can also use the :selected and :checked selectors to get at values, for example:

```
$('select.foo option:selected').val();  // get the value from a dropdown select
$('select.foo').val();          // get the value from a dropdown select even easier
$('input:checkbox:checked').val();    // get the value from a checked checkbox
$('input:radio[name=bar]:checked').val(); // get the value from a set of radio buttons
```

Note: At present, using .val() on textarea elements strips carriage return characters from the browser-reported value. When this value is sent to the server via XHR however, carriage returns are preserved (or added by browsers which do not include them in the raw value). A workaround for this issue can be achieved using a valHook as follows:

```
$.valHooks.textarea = {
    get: function( elem ) {
        return elem.value.replace( /r?n/g, "rn" );
    }
};
```

Example

Get the single value from a single select and an array of values from a multiple select and display their values.

```
function displayVals() {
   var singleValues = $("#single").val();
   var multipleValues = $("#multiple").val() || [];
   $("p").html("<b>Single:</b> " +
         singleValues +
         " <b>Multiple:</b> " +
         multipleValues.join(", "));
  }

  $("select").change(displayVals);
  displayVals();
```

Example

Find the value of an input box.

```
$("input").keyup(function () {
   var value = $(this).val();
   $("p").text(value);
  }).keyup();
```

val(value)

Set the value of each element in the set of matched elements.

Arguments

value - A string of text or an array of strings corresponding to the value of each matched element to set as selected/checked.

This method is typically used to set the values of form fields.

Passing an array of element values allows matching <input type="checkbox">, <input type="radio"> and <option>s inside of n <select multiple="multiple"> to be selected. In the case of <input type="radio">s that are part of a radio group and <select multiple="multiple"> the other elements will be deselected.

The .val() method allows us to set the value by passing in a function. As of jQuery 1.4, the function is passed two arguments, the current element's index and its current value:

```
$('input:text.items').val(function( index, value ) {
 return value + ' ' + this.className;
```

```
});
```

This example appends the string " items" to the text inputs' values.

Example
Set the value of an input box.

```
$("button").click(function () {
   var text = $(this).text();
   $("input").val(text);
  });
```

Example
Use the function argument to modify the value of an input box.

```
$('input').bind('blur', function() {
  $(this).val(function( i, val ) {
   return val.toUpperCase();
  });
 });
```

Example
Set a single select, a multiple select, checkboxes and a radio button .

```
$("#single").val("Single2");
  $("#multiple").val(["Multiple2", "Multiple3"]);
  $("input").val(["check1","check2", "radio1" ]);
```

text()
Get the combined text contents of each element in the set of matched elements, including their descendants.

Unlike the `.html()` method, `.text()` can be used in both XML and HTML documents. The result of the `.text()` method is a string containing the combined text of all matched elements. (Due to variations in the HTML parsers in different browsers, the text returned may vary in newlines and other white space.) Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
 <ul>
 <li>list item 1</li>
 <li>list <strong>item</strong> 2</li>
 </ul>
 </div>
```

The code `$('div.demo-container').text()` would produce the following result:

```
Demonstration Box list item 1 list item 2
```

The `.text()` method cannot be used on form inputs or scripts. To set or get the text value of `input` or `textarea` elements, use the .val() method. To get the value of a script element, use the .html() method.

As of jQuery 1.4, the `.text()` method returns the value of text and CDATA nodes as well as element nodes.

Example
Find the text in the first paragraph (stripping out the html), then set the html of the last paragraph to show it is just text (the red bold is gone).

```
var str = $("p:first").text();
  $("p:last").html(str);
```

text(textString)
Set the content of each element in the set of matched elements to the specified text.

Arguments
textString - A string of text to set as the content of each matched element.

Unlike the `.html()` method, `.text()` can be used in both XML and HTML documents.

We need to be aware that this method escapes the string provided as necessary so that it will render correctly in HTML. To do so, it calls the DOM method `.createTextNode()`, which replaces special characters with their HTML entity equivalents (such as `<` for <). Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
 <ul>
  <li>list item 1</li>
  <li>list <strong>item</strong> 2</li>
 </ul>
</div>
```

The code `$('div.demo-container').text('<p>This is a test.</p>');` will produce the following DOM output:

```
<div class="demo-container">
&lt;p&gt;This is a test.&lt;/p&gt;
</div>
```

It will appear on a rendered page as though the tags were exposed, like this:

```
<p>This is a test</p>
```

The `.text()` method cannot be used on input elements. For input field text, use the .val() method.

As of jQuery 1.4, the `.text()` method allows us to set the text content by passing in a function.

```
$('ul li').text(function(index) {
 return 'item number ' + (index + 1);
});
```

Given an unordered list with three `<li>` elements, this example will produce the following DOM output:

```
<ul>
 <li>item number 1</li>
 <li>item number 2</li>
 <li>item number 3</li>
</ul>
```

Example
Add text to the paragraph (notice the bold tag is escaped).

```
$("p").text("<b>Some</b> new text.");
```

html()
Get the HTML contents of the first element in the set of matched elements.

This method is not available on XML documents.

In an HTML document, `.html()` can be used to get the contents of any element. If the selector expression matches more than one element, only the first match will have its HTML content returned. Consider this code:

```
$('div.demo-container').html();
```

In order for the following 's content to be retrieved, it would have to be the first one with class="demo-container" in the document:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
</div>
```

The result would look like this:

```
<div class="demo-box">Demonstration Box</div>
```

This method uses the browser's innerHTML property. Some browsers may not return HTML that exactly replicates the HTML source in an original document. For example, Internet Explorer sometimes leaves off the quotes around attribute values if they contain only alphanumeric characters.

Example

Click a paragraph to convert it from html to text.

```
$("p").click(function () {
   var htmlStr = $(this).html();
   $(this).text(htmlStr);
  });
```

html(htmlString)

Set the HTML contents of each element in the set of matched elements.

Arguments

htmlString - A string of HTML to set as the content of each matched element.

The .html() method is not available in XML documents.

When .html() is used to set an element's content, any content that was in that element is completely replaced by the new content. Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
</div>
```

The content of can be set like this:

```
$('div.demo-container')
 .html('<p>All new content. <em>You bet!</em></p>');
```

That line of code will replace everything inside :

```
<div class="demo-container">
 <p>All new content. <em>You bet!</em></p>
</div>
```

As of jQuery 1.4, the .html() method allows the HTML content to be set by passing in a function.

```
$('div.demo-container').html(function() {
 var emph = '<em>' + $('p').length + ' paragraphs!</em>';
 return '<p>All new content for ' + emph + '</p>';
});
```

Given a document with six paragraphs, this example will set the HTML of to All new content for 6 paragraphs! .

This method uses the browser's innerHTML property. Some browsers may not generate a DOM that exactly replicates the HTML source provided. For example, Internet Explorer prior to version 8 will convert all href properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate compatibility layer.

Note: In Internet Explorer up to and including version 9, setting the text content of an HTML element may corrupt the text nodes of its children that are being removed from the document as a result of the operation. If you are keeping references to these DOM elements and need them to be unchanged, use `.empty().html(string)` instead of `.html(string)` so that the elements are removed from the document before the new string is assigned to the element.

Example
Add some html to each div.

```
$("div").html("<span class='red'>Hello <b>Again</b></span>");
```

Example
Add some html to each div then immediately do further manipulations to the inserted html.

```
$("div").html("<b>Wow!</b> Such excitement...");
  $("div b").append(document.createTextNode("!!!"))
       .css("color", "red");
```

is(selector)
Check the current matched set of elements against a selector, element, or jQuery object and return `true` if at least one of these elements matches the given arguments.

Arguments
selector - A string containing a selector expression to match elements against.

Unlike other filtering methods, `.is()` does not create a new jQuery object. Instead, it allows you to test the contents of a jQuery object without modification. This is often useful inside callbacks, such as event handlers.

Suppose you have a list, with two of its items containing a child element:

```
<ul>
 <li>list <strong>item 1</strong></li>
 <li><span>list item 2</span></li>
 <li>list item 3</li>
</ul>
```

You can attach a click handler to the <ul> element, and then limit the code to be triggered only when a list item itself, not one of its children, is clicked:

```
$("ul").click(function(event) {
 var $target = $(event.target);
 if ( $target.is("li") ) {
  $target.css("background-color", "red");
 }
});
```

Now, when the user clicks on the word "list" in the first item or anywhere in the third item, the clicked list item will be given a red background. However, when the user clicks on item 1 in the first item or anywhere in the second item, nothing will occur, because in those cases the target of the event would be `<strong>` or `<span>`, respectively.

Prior to jQuery 1.7, in selector strings with positional selectors such as `:first`, `:gt()`, or `:even`, the positional filtering is done against the jQuery object passed to `.is()`, *not* against the containing document. So for the HTML shown above, an expression such as `$("li:first").is("li:last")` returns `true`, but `$("li:first-child").is("li:last-child")` returns `false`. In addition, a bug in Sizzle prevented many positional selectors from working properly. These two factors made positional selectors almost unusable in filters.

Starting with jQuery 1.7, selector strings with positional selectors apply the selector against the document, and then determine whether the first element of the current jQuery set matches any of the resulting elements. So for the HTML shown above, an expression such as `$("li:first").is("li:last")` returns `false`. Note that since positional selectors are jQuery additions and not W3C standard, we recommend using the W3C selectors whenever feasible.

Using a Function

The second form of this method evaluates expressions related to elements based on a function rather than a selector. For each element, if the function returns `true`, `.is()` returns `true` as well. For example, given a somewhat more involved HTML snippet:

```
<ul>
 <li><strong>list</strong> item 1 - one strong tag</li>
 <li><strong>list</strong> item <strong>2</strong> -
  two <span>strong tags</span></li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

You can attach a click handler to every `<li>` that evaluates the number of `<strong>` elements within the clicked `<li>` at that time like so:

```
$("li").click(function() {
 var $li = $(this),
  isWithTwo = $li.is(function() {
   return $('strong', this).length === 2;
  });
 if ( isWithTwo ) {
  $li.css("background-color", "green");
 } else {
  $li.css("background-color", "red");
 }
});
```

Example

Shows a few ways is() can be used inside an event handler.

```
$("div").one('click', function () {
  if ($(this).is(":first-child")) {
   $("p").text("It's the first div.");
  } else if ($(this).is(".blue,.red")) {
   $("p").text("It's a blue or red div.");
  } else if ($(this).is(":contains('Peter')")) {
   $("p").text("It's Peter!");
  } else {
   $("p").html("It's nothing <em>special</em>.");
  }
  $("p").hide().slideDown("slow");
  $(this).css({"border-style": "inset", cursor:"default"});
 });
```

Example

Returns true, because the parent of the input is a form element.

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
 $("div").text("isFormParent = " + isFormParent);
```

Example

Returns false, because the parent of the input is a p element.

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
 $("div").text("isFormParent = " + isFormParent);
```

Example

Checks against an existing collection of alternating list elements. Blue, alternating list elements slide up while others turn red.

```
var $alt = $("#browsers li:nth-child(2n)").css("background", "#00FFFF");
```

```
 $('li').click(function() {
  var $li = $(this);
  if ( $li.is( $alt ) ) {
   $li.slideUp();
  } else {
   $li.css("background", "red");
  }
 });
```

Example

An alternate way to achieve the above example using an element rather than a jQuery object. Checks against an existing collection of alternating list elements. Blue, alternating list elements slide up while others turn red.

```
var $alt = $("#browsers li:nth-child(2n)").css("background", "#00FFFF");
 $('li').click(function() {
  if ( $alt.is( this ) ) {
   $(this).slideUp();
  } else {
   $(this).css("background", "red");
  }
 });
```

filter(selector)

Reduce the set of matched elements to those that match the selector or pass the function's test.

Arguments

selector - A string containing a selector expression to match the current set of elements against.

Given a jQuery object that represents a set of DOM elements, the `.filter()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; all elements matching the selector will be included in the result.

Consider a page with a simple list on it: list item 1 list item 2 list item 3 list item 4 list item 5 list item 6

We can apply this method to the set of list items:

```
 $('li').filter(':even').css('background-color', 'red');
```

The result of this call is a red background for items 1, 3, and 5, as they match the selector (recall that `:even` and `:odd` use 0-based indexing).

Using a Filter Function

The second form of this method allows us to filter elements against a function rather than a selector. For each element, if the function returns `true` (or a "truthy" value), the element will be included in the filtered set; otherwise, it will be excluded. Suppose we have a somewhat more involved HTML snippet:

```
<ul>
 <li><strong>list</strong> item 1 -
  one strong tag</li>
 <li><strong>list</strong> item <strong>2</strong> -
  two <span>strong tags</span></li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
 <li>list item 6</li>
</ul>
```

We can select the list items, then filter them based on their contents:

```
$('li').filter(function(index) {
 return $('strong', this).length == 1;
}).css('background-color', 'red');
```

This code will alter the first list item only, as it contains exactly one `<strong>` tag. Within the filter function, `this` refers to each DOM element in turn. The parameter passed to the function tells us the index of that DOM element within the set matched by the jQuery object.

We can also take advantage of the `index` passed through the function, which indicates the 0-based position of the element within the unfiltered set of matched elements:

```
$('li').filter(function(index) {
 return index % 3 == 2;
}).css('background-color', 'red');
```

This alteration to the code will cause the third and sixth list items to be highlighted, as it uses the modulus operator (`%`) to select every item with an `index` value that, when divided by 3, has a remainder of 2.

Example
Change the color of all divs; then add a border to those with a "middle" class.

```
$("div").css("background", "#c8ebcc")
      .filter(".middle")
      .css("border-color", "red");
```

Example
Change the color of all divs; then add a border to the second one (index == 1) and the div with an id of "fourth."

```
$("div").css("background", "#b4b0da")
      .filter(function (index) {
         return index == 1 || $(this).attr("id") == "fourth";
        })
      .css("border", "3px double red");
```

Example
Select all divs and filter the selection with a DOM element, keeping only the one with an id of "unique".

```
$("div").filter( document.getElementById("unique") )
```

Example
Select all divs and filter the selection with a jQuery object, keeping only the one with an id of "unique".

```
$("div").filter( $("#unique") )
```

toggleClass(className)
Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Arguments
className - One or more class names (separated by spaces) to be toggled for each element in the matched set.

This method takes one or more class names as its parameter. In the first version, if an element in the matched set of elements already has the class, then it is removed; if an element does not have the class, then it is added. For example, we can apply `.toggleClass()` to a simple `<div>`:

```
<div class="tumble">Some text.</div>
```

The first time we apply `$('div.tumble').toggleClass('bounce')`, we get the following:

```
<div class="tumble bounce">Some text.</div>
```

The second time we apply `$('div.tumble').toggleClass('bounce')`, the `<div>` class is returned to the single `tumble` value:

```
<div class="tumble">Some text.</div>
```

Applying `.toggleClass('bounce spin')` to the same `<div>` alternates between `<div class="tumble bounce spin">` and `<div class="tumble">`.

The second version of `.toggleClass()` uses the second parameter for determining whether the class should be added or removed. If this parameter's value is `true`, then the class is added; if `false`, the class is removed. In essence, the statement:

```
$('#foo').toggleClass(className, addOrRemove);
```

is equivalent to:

```
if (addOrRemove) {
  $('#foo').addClass(className);
 }
 else {
  $('#foo').removeClass(className);
 }
```

As of jQuery 1.4, if no arguments are passed to `.toggleClass()`, all class names on the element the first time `.toggleClass()` is called will be toggled. Also as of jQuery 1.4, the class name to be toggled can be determined by passing in a function.

```
$('div.foo').toggleClass(function() {
 if ($(this).parent().is('.bar')) {
  return 'happy';
 } else {
  return 'sad';
 }
});
```

This example will toggle the `happy` class for `<div class="foo">` elements if their parent element has a class of `bar`; otherwise, it will toggle the `sad` class.

Example

Toggle the class 'highlight' when a paragraph is clicked.

```
$("p").click(function () {
   $(this).toggleClass("highlight");
  });
```

Example

Add the "highlight" class to the clicked paragraph on every third click of that paragraph, remove it every first and second click.

```
var count = 0;
$("p").each(function() {
 var $thisParagraph = $(this);
 var count = 0;
 $thisParagraph.click(function() {
  count++;
  $thisParagraph.find("span").text('clicks: ' + count);
  $thisParagraph.toggleClass("highlight", count % 3 == 0);
 });
});
```

Example

Toggle the class name(s) indicated on the buttons for each div.

```
var cls = ['', 'a', 'a b', 'a b c'];
var divs = $('div.wrap').children();
var appendClass = function() {
 divs.append(function() {
  return '<div>' + (this.className || 'none') + '</div>';
 });
};

appendClass();

$('button').bind('click', function() {
 var tc = this.className || undefined;
 divs.toggleClass(tc);
 appendClass();
});

$('a').bind('click', function(event) {
 event.preventDefault();
 divs.empty().each(function(i) {
  this.className = cls[i];
 });
 appendClass();
});
```

removeClass([className])

Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Arguments

className - One or more space-separated classes to be removed from the class attribute of each matched element.

If a class name is included as a parameter, then only that class will be removed from the set of matched elements. If no class names are specified in the parameter, all classes will be removed.

More than one class may be removed at a time, separated by a space, from the set of matched elements, like so:

```
$('p').removeClass('myClass yourClass')
```

This method is often used with `.addClass()` to switch elements' classes from one to another, like so:

```
$('p').removeClass('myClass noClass').addClass('yourClass');
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

To replace all existing classes with another class, we can use `.attr('class', 'newClass')` instead.

As of jQuery 1.4, the `.removeClass()` method allows us to indicate the class to be removed by passing in a function.

```
$('li:last').removeClass(function() {
     return $(this).prev().attr('class');
    });
```

This example removes the class name of the penultimate `<li>` from the last `<li>`.

Example

Remove the class 'blue' from the matched elements.

```
$("p:even").removeClass("blue");
```

Example
Remove the class 'blue' and 'under' from the matched elements.

```
$("p:odd").removeClass("blue under");
```

Example
Remove all the classes from the matched elements.

```
$("p:eq(1)").removeClass();
```

removeAttr(attributeName)
Remove an attribute from each element in the set of matched elements.

Arguments
attributeName - An attribute to remove; as of version 1.7, it can be a space-separated list of attributes.

The `.removeAttr()` method uses the JavaScript `removeAttribute()` function, but it has the advantage of being able to be called directly on a jQuery object and it accounts for different attribute naming across browsers.

Note: Removing an inline `onclick` event handler using `.removeAttr()` doesn't achieve the desired effect in Internet Explorer 6, 7, or 8. To avoid potential problems, use `.prop()` instead:

```
$element.prop("onclick", null);
console.log("onclick property: ", $element[0].onclick);
```

Example
Clicking the button enables the input next to it.

```
(function() {
 var inputTitle = $("input").attr("title");
 $("button").click(function () {
  var input = $(this).next();

  if ( input.attr("title") == inputTitle ) {
   input.removeAttr("title")
  } else {
   input.attr("title", inputTitle);
  }

  $("#log").html( "input title is now " + input.attr("title") );
 });
})();
```

attr(attributeName)
Get the value of an attribute for the first element in the set of matched elements.

Arguments
attributeName - The name of the attribute to get.

The `.attr()` method gets the attribute value for only the *first* element in the matched set. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

As of jQuery 1.6, the `.attr()` method returns `undefined` for attributes that have not been set. In addition, `.attr()` should not be used on plain objects, arrays, the window, or the document. To retrieve and change DOM properties, use the .prop() method.

Using jQuery's `.attr()` method to get the value of an element's attribute has two main benefits:
- **Convenience**: It can be called directly on a jQuery object and chained to other jQuery methods.
- **Cross-browser consistency**: The values of some attributes are reported inconsistently across browsers, and even across versions of a single browser. The `.attr()` method reduces such inconsistencies.

Note: Attribute values are strings with the exception of a few attributes such as value and tabindex.

Example
Find the title attribute of the first in the page.

```
var title = $("em").attr("title");
 $("div").text(title);
```

attr(attributeName, value)
Set one or more attributes for the set of matched elements.

Arguments
attributeName - The name of the attribute to set.
value - A value to set for the attribute.

The `.attr()` method is a convenient way to set the value of attributes-especially when setting multiple attributes or using values returned by a function. Consider the following image:

```
<img id="greatphoto" src="brush-seller.jpg" alt="brush seller" />
```

Setting a simple attribute

To change the `alt` attribute, simply pass the name of the attribute and its new value to the `.attr()` method:

```
$('#greatphoto').attr('alt', 'Beijing Brush Seller');
```

Add an attribute the same way:

```
$('#greatphoto')
.attr('title', 'Photo by Kelly Clark');
```

Setting several attributes at once

To change the `alt` attribute and add the `title` attribute at the same time, pass both sets of names and values into the method at once using a map (JavaScript object literal). Each key-value pair in the map adds or modifies an attribute:

```
$('#greatphoto').attr({
 alt: 'Beijing Brush Seller',
 title: 'photo by Kelly Clark'
});
```

When setting multiple attributes, the quotes around attribute names are optional.

WARNING: When setting the 'class' attribute, you must always use quotes!

Note: jQuery prohibits changing the `type` attribute on an `<input>` or `<button>` element and will throw an error in all browsers. This is because the `type` attribute cannot be changed in Internet Explorer.

Computed attribute values

By using a function to set attributes, you can compute the value based on other properties of the element. For example, to concatenate a new value with an existing value:

```
$('#greatphoto').attr('title', function(i, val) {
 return val + ' - photo by Kelly Clark'
});
```

This use of a function to compute attribute values can be particularly useful when modifying the attributes of multiple elements at once.

Note: If nothing is returned in the setter function (ie. `function(index, attr){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example
Set some attributes for all <img>s in the page.

```
$("img").attr({
 src: "/images/hat.gif",
 title: "jQuery",
 alt: "jQuery Logo"
});
$("div").text($("img").attr("alt"));
```

Example
Set the id for divs based on the position in the page.

```
$("div").attr("id", function (arr) {
 return "div-id" + arr;
})
.each(function () {
 $("span", this).html("(ID = '<b>" + this.id + "</b>')");
});
```

Example
Set the src attribute from title attribute on the image.

```
$("img").attr("src", function() {
  return "/images/" + this.title;
});
```

addClass(className)
Adds the specified class(es) to each of the set of matched elements.

Arguments
className - One or more class names to be added to the class attribute of each matched element.

It's important to note that this method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.

More than one class may be added at a time, separated by a space, to the set of matched elements, like so:

```
$("p").addClass("myClass yourClass");
```

This method is often used with `.removeClass()` to switch elements' classes from one to another, like so:

```
$("p").removeClass("myClass noClass").addClass("yourClass");
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

As of jQuery 1.4, the `.addClass()` method's argument can receive a function.

```
$("ul li:last").addClass(function(index) {
 return "item-" + index;
});
```

Given an unordered list with five `<li>` elements, this example adds the class "item-4" to the last `<li>`.

Example
Adds the class "selected" to the matched elements.

```
$("p:last").addClass("selected");
```

Example
Adds the classes "selected" and "highlight" to the matched elements.

```
$("p:last").addClass("selected highlight");
```

Example
Pass in a function to `.addClass()` to add the "green" class to a div that already has a "red" class.

```
$("div").addClass(function(index, currentClass) {
  var addedClass;

  if ( currentClass === "red" ) {
   addedClass = "green";
   $("p").text("There is one green div");
  }

  return addedClass;
 });
```

Version 1.0.4

event.pageY
The mouse position relative to the top edge of the document.

Example
Show the mouse position relative to the left and top edges of the document (within this iframe).

```
$(document).bind('mousemove',function(e){
      $("#log").text("e.pageX: " + e.pageX + ", e.pageY: " + e.pageY);
});
```

event.pageX
The mouse position relative to the left edge of the document.

Example
Show the mouse position relative to the left and top edges of the document (within the iframe).

```
$(document).bind('mousemove',function(e){
      $("#log").text("e.pageX: " + e.pageX + ", e.pageY: " + e.pageY);
});
```

jQuery.globalEval(code)
Execute some JavaScript code globally.

Arguments
code - The JavaScript code to execute.

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).

Example
Execute a script in the global context.

```
function test(){
  jQuery.globalEval("var newVar = true;")
}
test();
```

```
// newVar === true
```

Version 1.1

event.data
An optional data map passed to an event method when the current executing handler is bound.

Example
Within a `for` loop, pass the value of `i` to the `.on()` method so that the current iteration's value is preserved.

```
var logDiv = $("#log");

/* Note: This code is for demonstration purposes only. */
for (var i = 0; i < 5; i++) {
 $("button").eq(i).on("click", {value: i}, function(event) {
  var msgs = [
   "button = " + $(this).index(),
   "event.data.value = " + event.data.value,
   "i = " + i
  ];
  logDiv.append( msgs.join(", ") + "<br>" );
 });
}
```

one(events, [data], handler(eventObject))
Attach a handler to an event for the elements. The handler is executed at most once per element.

Arguments
events - A string containing one or more JavaScript event types, such as "click" or "submit," or custom event names.
data - A map of data that will be passed to the event handler.
handler(eventObject) - A function to execute at the time the event is triggered.

The first form of this method is identical to `.bind()`, except that the handler is unbound after its first invocation. The second two forms, introduced in jQuery 1.7, are identical to `.on()` except that the handler is removed after the first time the event occurs at the delegated element, whether the selector matched anything or not. For example:

```
$("#foo").one("click", function() {
 alert("This will be displayed only once.");
});
$("body").one("click", "#foo", function() {
 alert("This displays if #foo is the first thing clicked in the body.");
});
```

After the code is executed, a click on the element with ID `foo` will display the alert. Subsequent clicks will do nothing. This code is equivalent to:

```
$("#foo").bind("click", function( event ) {
 alert("This will be displayed only once.");
 $(this).unbind( event );
});
```

In other words, explicitly calling `.unbind()` from within a regularly-bound handler has exactly the same effect.

If the first argument contains more than one space-separated event types, the event handler is called *once for each event type*.

Example
Tie a one-time click to each div.

```
var n = 0;
$("div").one("click", function() {
 var index = $("div").index(this);
 $(this).css({
  borderStyle:"inset",
  cursor:"auto"
 });
 $("p").text("Div at index #" + index + " clicked." +
   " That's " + ++n + " total clicks.");
});
```

Example
To display the text of all paragraphs in an alert box the first time each of them is clicked:

```
$("p").one("click", function(){
alert( $(this).text() );
});
```

jQuery.ajaxSetup(options)
Set default values for future Ajax requests.

Arguments
options - A set of key/value pairs that configure the default Ajax request. All options are optional.

For details on the settings available for `$.ajaxSetup()`, see `$.ajax()`.

All subsequent Ajax calls using any function will use the new settings, unless overridden by the individual calls, until the next invocation of `$.ajaxSetup()`.

For example, the following sets a default for the `url` parameter before pinging the server repeatedly:

```
$.ajaxSetup({
 url: 'ping.php'
});
```

Now each time an Ajax request is made, the "ping.php" URL will be used automatically:

```
$.ajax({
 // url not set here; uses ping.php
 data: {'name': 'Dan'}
});
```

Note: Global callback functions should be set with their respective global Ajax event handler methods-`.ajaxStart()`, `.ajaxStop()`, `.ajaxComplete()`, `.ajaxError()`, `.ajaxSuccess()`, `.ajaxSend()`-rather than within the `options` object for `$.ajaxSetup()`.

Example
Sets the defaults for Ajax requests to the url "/xmlhttp/", disables global handlers and uses POST instead of GET. The following Ajax requests then sends some data without having to set anything else.

```
$.ajaxSetup({
  url: "/xmlhttp/",
  global: false,
  type: "POST"

 });
 $.ajax({ data: myData });
```

attr(attributeName)
Get the value of an attribute for the first element in the set of matched elements.

Arguments

attributeName - The name of the attribute to get.

The `.attr()` method gets the attribute value for only the *first* element in the matched set. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

As of jQuery 1.6, the `.attr()` method returns `undefined` for attributes that have not been set. In addition, `.attr()` should not be used on plain objects, arrays, the window, or the document. To retrieve and change DOM properties, use the .prop() method.

Using jQuery's `.attr()` method to get the value of an element's attribute has two main benefits:

- **Convenience**: It can be called directly on a jQuery object and chained to other jQuery methods.
- **Cross-browser consistency**: The values of some attributes are reported inconsistently across browsers, and even across versions of a single browser. The `.attr()` method reduces such inconsistencies.

Note: Attribute values are strings with the exception of a few attributes such as value and tabindex.

Example

Find the title attribute of the first in the page.

```
var title = $("em").attr("title");
 $("div").text(title);
```

attr(attributeName, value)

Set one or more attributes for the set of matched elements.

Arguments

attributeName - The name of the attribute to set.
value - A value to set for the attribute.

The `.attr()` method is a convenient way to set the value of attributes-especially when setting multiple attributes or using values returned by a function. Consider the following image:

```
<img id="greatphoto" src="brush-seller.jpg" alt="brush seller" />
```

Setting a simple attribute

To change the `alt` attribute, simply pass the name of the attribute and its new value to the `.attr()` method:

```
$('#greatphoto').attr('alt', 'Beijing Brush Seller');
```

Add an attribute the same way:

```
$('#greatphoto')
.attr('title', 'Photo by Kelly Clark');
```

Setting several attributes at once

To change the `alt` attribute and add the `title` attribute at the same time, pass both sets of names and values into the method at once using a map (JavaScript object literal). Each key-value pair in the map adds or modifies an attribute:

```
$('#greatphoto').attr({
 alt: 'Beijing Brush Seller',
 title: 'photo by Kelly Clark'
});
```

When setting multiple attributes, the quotes around attribute names are optional.

WARNING: When setting the 'class' attribute, you must always use quotes!

Note: jQuery prohibits changing the `type` attribute on an `<input>` or `<button>` element and will throw an error in all browsers. This is because the `type` attribute cannot be changed in Internet Explorer.

Computed attribute values

By using a function to set attributes, you can compute the value based on other properties of the element. For example, to concatenate a new value with an existing value:

```
$('#greatphoto').attr('title', function(i, val) {
 return val + ' - photo by Kelly Clark'
});
```

This use of a function to compute attribute values can be particularly useful when modifying the attributes of multiple elements at once.

Note: If nothing is returned in the setter function (ie. `function(index, attr){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example
Set some attributes for all <img>s in the page.

```
$("img").attr({
 src: "/images/hat.gif",
 title: "jQuery",
 alt: "jQuery Logo"
});
$("div").text($("img").attr("alt"));
```

Example
Set the id for divs based on the position in the page.

```
$("div").attr("id", function (arr) {
 return "div-id" + arr;
})
.each(function () {
 $("span", this).html("(ID = '<b>" + this.id + "</b>')");
});
```

Example
Set the src attribute from title attribute on the image.

```
$("img").attr("src", function() {
  return "/images/" + this.title;
});
```

Version 1.1.2

eq(index)
Reduce the set of matched elements to the one at the specified index.

Arguments
index - An integer indicating the 0-based position of the element.

Given a jQuery object that represents a set of DOM elements, the `.eq()` method constructs a new jQuery object from one element within that set. The supplied index identifies the position of this element in the set.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
```

```
  <li>list item 5</li>
 </ul>
```

We can apply this method to the set of list items:

```
 $('li').eq(2).css('background-color', 'red');
```

The result of this call is a red background for item 3. Note that the supplied index is zero-based, and refers to the position of the element within the jQuery object, not within the DOM tree.

Providing a negative number indicates a position starting from the end of the set, rather than the beginning. For example:

```
 $('li').eq(-2).css('background-color', 'red');
```

This time list item 4 is turned red, since it is two from the end of the set.

If an element cannot be found at the specified zero-based index, the method constructs a new jQuery object with an empty set and a `length` property of 0.

```
 $('li').eq(5).css('background-color', 'red');
```

Here, none of the list items is turned red, since `.eq(5)` indicates the sixth of five list items.

Example
Turn the div with index 2 blue by adding an appropriate class.

```
$("body").find("div").eq(2).addClass("blue");
```

Version 1.1.3

event.which
For key or mouse events, this property indicates the specific key or button that was pressed.

The `event.which` property normalizes `event.keyCode` and `event.charCode`. It is recommended to watch `event.which` for keyboard key input. For more detail, read about event.charCode on the MDC.

`event.which` also normalizes button presses (`mousedown` and `mouseupe`vents), reporting `1` for left button, `2` for middle, and `3` for right. Use `event.which` instead of `event.button`.

Example
Log which key was depressed.

```
$('#whichkey').bind('keydown',function(e){
 $('#log').html(e.type + ': ' + e.which );
});
```

Example
Log which mouse button was depressed.

```
$('#whichkey').bind('mousedown',function(e){
 $('#log').html(e.type + ': ' + e.which );
});
```

jQuery.browser

Contains flags for the useragent, read from navigator.userAgent. **We recommend against using this property; please try to use feature detection instead (see jQuery.support). jQuery.browser may be moved to a plugin in a future release of jQuery.**

The `$.browser` property provides information about the web browser that is accessing the page, as reported by the browser itself. It contains flags for each of the four most prevalent browser classes (Internet Explorer, Mozilla, Webkit, and Opera) as well as version information.

Available flags are:

- webkit (as of jQuery 1.4)
- safari (deprecated)
- opera
- msie
- mozilla

This property is available immediately. It is therefore safe to use it to determine whether or not to call `$(document).ready()`.The `$.browser` property is deprecated in jQuery 1.3, and its functionality may be moved to a team-supported plugin in a future release of jQuery.

Because `$.browser` uses `navigator.userAgent` to determine the platform, it is vulnerable to spoofing by the user or misrepresentation by the browser itself. It is always best to avoid browser-specific code entirely where possible. The `$.support` property is available for detection of support for particular features rather than relying on `$.browser`.

Example

Show the browser info.

```
jQuery.each(jQuery.browser, function(i, val) {
   $("<div>" + i + " : <span>" + val + "</span>")
        .appendTo( document.body );
  });
```

Example

Returns true if the current useragent is some version of Microsoft's Internet Explorer.

```
$.browser.msie;
```

Example

Alerts "this is WebKit!" only for WebKit browsers

```
if ($.browser.webkit) {
  alert( "this is webkit!" );
 }
```

Example

Alerts "Do stuff for Firefox 3" only for Firefox 3 browsers.

```
var ua = $.browser;
 if ( ua.mozilla && ua.version.slice(0,3) == "1.9" ) {
  alert( "Do stuff for firefox 3" );
 }
```

Example

Set a CSS property that's specific to a particular browser.

```
if ( $.browser.msie ) {
  $("#div ul li").css( "display","inline" );
 } else {
  $("#div ul li").css( "display","inline-table" );
 }
```

jQuery.browser.version

The version number of the rendering engine for the user's browser.

Here are some typical results:

- Internet Explorer: 6.0, 7.0, 8.0
- Mozilla/Firefox/Flock/Camino: 1.7.12, 1.8.1.3, 1.9
- Opera: 10.06, 11.01
- Safari/Webkit: 312.8, 418.9

Note that IE8 claims to be 7 in Compatibility View.

Example

Returns the version number of the rendering engine used by the user's current browser. For example, FireFox 4 returns 2.0 (the version of the Gecko rendering engine it utilizes).

```
$("p").html( "The version number of the rendering engine your browser uses is: <span>" +
        $.browser.version + "</span>" );
```

Example

Alerts the version of IE's rendering engine that is being used:

```
if ( $.browser.msie ) {
 alert( $.browser.version );
}
```

Example

Often you only care about the "major number," the whole number, which you can get by using JavaScript's built-in `parseInt()` function:

```
if ( $.browser.msie ) {
 alert( parseInt($.browser.version, 10) );
}
```

jQuery.unique(array)

Sorts an array of DOM elements, in place, with the duplicates removed. Note that this only works on arrays of DOM elements, not strings or numbers.

Arguments

array - The Array of DOM elements.

The `$.unique()` function searches through an array of objects, sorting the array, and removing any duplicate nodes. This function only works on plain JavaScript arrays of DOM elements, and is chiefly used internally by jQuery.

As of jQuery 1.4 the results will always be returned in document order.

Example

Removes any duplicate elements from the array of divs.

```
var divs = $("div").get(); // unique() must take a native array

  // add 3 elements of class dup too (they are divs)
  divs = divs.concat($(".dup").get());
  $("div:eq(1)").text("Pre-unique there are " + divs.length + " elements.");

  divs = jQuery.unique(divs);
  $("div:eq(2)").text("Post-unique there are " + divs.length + " elements.")
         .css("color", "red");
```

Version 1.1.4

event.relatedTarget

The other DOM element involved in the event, if any.

For `mouseout`, indicates the element being entered; for `mouseover`, indicates the element being exited.

Example
On mouseout of anchors, alert the element type being entered.

```
$("a").mouseout(function(event) {
 alert(event.relatedTarget.nodeName); // "DIV"
});
```

jQuery.isXMLDoc(node)
Check to see if a DOM node is within an XML document (or is an XML document).

Arguments
node - The DOM node that will be checked to see if it's in an XML document.

Example
Check an object to see if it's in an XML document.

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

only-child
Selects all elements that are the only child of their parent.

If the parent has other child elements, nothing is matched.

Example
Change the text and add a border for each button that is the only child of its parent.

```
$("div button:only-child").text("Alone").css("border", "2px blue solid");
```

last-child
Selects all elements that are the last child of their parent.

While :last matches only a single element, `:last-child` can match more than one: one for each parent.

Example
Finds the last span in each matched div and adds some css plus a hover state.

```
$("div span:last-child")
    .css({color:"red", fontSize:"80%"})
    .hover(function () {
       $(this).addClass("solast");
      }, function () {
       $(this).removeClass("solast");
      });
```

first-child
Selects all elements that are the first child of their parent.

While :first matches only a single element, the `:first-child` selector can match more than one: one for each parent. This is equivalent to `:nth-child(1)`.

Example
Finds the first span in each matched div to underline and add a hover state.

```
$("div span:first-child")
    .css("text-decoration", "underline")
```

```
    .hover(function () {
       $(this).addClass("sogreen");
      }, function () {
       $(this).removeClass("sogreen");
      });
```

nth-child
Selects all elements that are the nth-child of their parent.

Arguments
index - The index of each child to match, starting with `1`, the string `even` or `odd`, or an equation (eg. `:nth-child(even)`, `:nth-child(4n)`)

Because jQuery's implementation of `:nth-child(n)` is strictly derived from the CSS specification, the value of n is "1-indexed", meaning that the counting starts at 1. For all other selector expressions, however, jQuery follows JavaScript's "0-indexed" counting. Therefore, given a single `<ul>` containing two `<li>`s, `$('li:nth-child(1)')` selects the first `<li>` while `$('li:eq(1)')` selects the second.

The `:nth-child(n)` pseudo-class is easily confused with `:eq(n)`, even though the two can result in dramatically different matched elements. With `:nth-child(n)`, all children are counted, regardless of what they are, and the specified element is selected only if it matches the selector attached to the pseudo-class. With `:eq(n)` only the selector attached to the pseudo-class is counted, not limited to children of any other element, and the (n+1)th one (n is 0-based) is selected.

Further discussion of this unusual usage can be found in the W3C CSS specification.

Example
Finds the second li in each matched ul and notes it.

```
$("ul li:nth-child(2)").append("<span> - 2nd!</span>");
```

Example
This is a playground to see how the selector works with different strings. Notice that this is different from the :even and :odd which have no regard for parent and just filter the list of elements to every other one. The :nth-child, however, counts the index of the child to its particular parent. In any case, it's easier to see than explain so...

```
$("button").click(function () {
   var str = $(this).text();
   $("tr").css("background", "white");
   $("tr" + str).css("background", "#ff0000");
   $("#inner").text(str);
  });
```

has
Selects elements which contain at least one element that matches the specified selector.

Arguments
selector - Any selector.

The expression `$('div:has(p)')` matches a `<div>` if a `<p>` exists anywhere among its descendants, not just as a direct child.

Example
Adds the class "test" to all divs that have a paragraph inside of them.

```
$("div:has(p)").addClass("test");
```

contains
Select all elements that contain the specified text.

Arguments
text - A string of text to look for. It's case sensitive.

The matching text can appear directly within the selected element, in any of that element's descendants, or a combination thereof. As with attribute

value selectors, text inside the parentheses of `:contains()` can be written as a bare word or surrounded by quotation marks. The text must have matching case to be selected.

Example
Finds all divs containing "John" and underlines them.

```
$("div:contains('John')").css("text-decoration", "underline");
```

slice(start, [end])
Reduce the set of matched elements to a subset specified by a range of indices.

Arguments
start - An integer indicating the 0-based position at which the elements begin to be selected. If negative, it indicates an offset from the end of the set.
end - An integer indicating the 0-based position at which the elements stop being selected. If negative, it indicates an offset from the end of the set. If omitted, the range continues until the end of the set.

Given a jQuery object that represents a set of DOM elements, the `.slice()` method constructs a new jQuery object from a subset of the matching elements. The supplied `start` index identifies the position of one of the elements in the set; if `end` is omitted, all elements after this one will be included in the result.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').slice(2).css('background-color', 'red');
```

The result of this call is a red background for items 3, 4, and 5. Note that the supplied index is zero-based, and refers to the position of elements within the jQuery object, not within the DOM tree.

The end parameter allows us to limit the selected range even further. For example:

```
$('li').slice(2, 4).css('background-color', 'red');
```

Now only items 3 and 4 are selected. The index is once again zero-based; the range extends up to but not including the specified index.

Negative Indices

The jQuery `.slice()` method is patterned after the JavaScript .slice() method for arrays. One of the features that it mimics is the ability for negative numbers to be passed as either the `start` or `end` parameter. If a negative number is provided, this indicates a position starting from the end of the set, rather than the beginning. For example:

```
$('li').slice(-2, -1).css('background-color', 'red');
```

This time only list item 4 is turned red, since it is the only item in the range between two from the end (`-2`) and one from the end (`-1`).

Example
Turns divs yellow based on a random slice.

```
function colorEm() {
   var $div = $("div");
   var start = Math.floor(Math.random() *
```

```
              $div.length);
  var end = Math.floor(Math.random() *
            ($div.length - start)) +
            start + 1;
  if (end == $div.length) end = undefined;
  $div.css("background", "");
  if (end)
   $div.slice(start, end).css("background", "yellow");
   else
   $div.slice(start).css("background", "yellow");

  $("span").text('$("div").slice(' + start +
          (end ? ', ' + end : '') +
          ').css("background", "yellow");');
}

$("button").click(colorEm);
```

Example
Selects all paragraphs, then slices the selection to include only the first element.

```
$("p").slice(0, 1).wrapInner("<b></b>");
```

Example
Selects all paragraphs, then slices the selection to include only the first and second element.

```
$("p").slice(0, 2).wrapInner("<b></b>");
```

Example
Selects all paragraphs, then slices the selection to include only the second element.

```
$("p").slice(1, 2).wrapInner("<b></b>");
```

Example
Selects all paragraphs, then slices the selection to include only the second and third element.

```
$("p").slice(1).wrapInner("<b></b>");
```

Example
Selects all paragraphs, then slices the selection to include only the third element.

```
$("p").slice(-1).wrapInner("<b></b>");
```

Version 1.2

animated
Select all elements that are in the progress of an animation at the time the selector is run.

Example
Change the color of any div that is animated.

```
$("#run").click(function(){
  $("div:animated").toggleClass("colored");
});
function animateIt() {
 $("#mover").slideToggle("slow", animateIt);
}
animateIt();
```

header
Selects all elements that are headers, like h1, h2, h3 and so on.

Example
Adds a background and text color to all the headers on the page.

```
$(":header").css({ background:'#CCC', color:'blue' });
```

dequeue([queueName])
Execute the next function on the queue for the matched elements.

Arguments
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

When `.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 `.dequeue()` to be called, so that the sequence can continue.

Example
Use dequeue to end a custom queue function which allows the queue to keep going.

```
$("button").click(function () {
 $("div").animate({left:'+=200px'}, 2000);
 $("div").animate({top:'0px'}, 600);
 $("div").queue(function () {
  $(this).toggleClass("red");
  $(this).dequeue();
 });
 $("div").animate({left:'10px', top:'30px'}, 700);
});
```

queue([queueName])
Show the queue of functions to be executed on the matched elements.

Arguments
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Example
Show the length of the queue.

```
var div = $("div");

function runIt() {
 div.show("slow");
 div.animate({left:'+=200'},2000);
 div.slideToggle(1000);
 div.slideToggle("fast");
 div.animate({left:'-=200'},1500);
 div.hide("slow");
 div.show(1200);
 div.slideUp("normal", runIt);
}

function showIt() {
 var n = div.queue("fx");
 $("span").text( n.length );
 setTimeout(showIt, 100);
}

runIt();
showIt();
```

queue([queueName], newQueue)

Manipulate the queue of functions to be executed on the matched elements.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.
newQueue - An array of functions to replace the current queue contents.

Every element can have one to many queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution. The typical example of this is calling multiple animation methods on an element. For example:

```
$('#foo').slideUp().fadeIn();
```

When this statement is executed, the element begins its sliding animation immediately, but the fading transition is placed on the `fx` queue to be called only once the sliding transition is complete.

The `.queue()` method allows us to directly manipulate this queue of functions. Calling `.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue.

This feature is similar to providing a callback function with an animation method, but does not require the callback to be given at the time the animation is performed.

```
$('#foo').slideUp();
$('#foo').queue(function() {
 alert('Animation complete.');
 $(this).dequeue();
});
```

This is equivalent to:

```
$('#foo').slideUp(function() {
 alert('Animation complete.');
});
```

Note that when adding a function with `.queue()`, we should ensure that `.dequeue()` is eventually called so that the next function in line executes.

As of jQuery 1.4, the function that's called is passed another function as the first argument. When called, this automatically dequeues the next item and keeps the queue moving. We use it as follows:

```
$("#test").queue(function(next) {
  // Do some stuff...
  next();
});
```

Example

Queue a custom function.

```
$(document.body).click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
   });
   $("div").animate({left:'-=200'},500);
   $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
   });
   $("div").slideUp();
  });
```

Example

Set a queue array to delete the queue.

```
$("#start").click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},5000);
   $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
   });
   $("div").animate({left:'-=200'},1500);
   $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
   });
   $("div").slideUp();
  });
  $("#stop").click(function () {
   $("div").queue("fx", []);
   $("div").stop();
  });
```

triggerHandler(eventType, [extraParameters])

Execute all handlers attached to an element for an event.

Arguments

eventType - A string containing a JavaScript event type, such as `click` or `submit`.
extraParameters - An array of additional parameters to pass along to the event handler.

The `.triggerHandler()` method behaves similarly to `.trigger()`, with the following exceptions:

- The `.triggerHandler()` method does not cause the default behavior of an event to occur (such as a form submission).
- While `.trigger()` will operate on all elements matched by the jQuery object, `.triggerHandler()` only affects the first matched element.
- Events created with `.triggerHandler()` do not bubble up the DOM hierarchy; if they are not handled by the target element directly, they do nothing.
- Instead of returning the jQuery object (to allow chaining), `.triggerHandler()` returns whatever value was returned by the last handler it caused to be executed. If no handlers are triggered, it returns `undefined`

For more information on this method, see the discussion for `.trigger()`.

Example

If you called .triggerHandler() on a focus event - the browser's default focus action would not be triggered, only the event handlers bound to the focus event.

```
$("#old").click(function(){
$("input").trigger("focus");
});
$("#new").click(function(){
$("input").triggerHandler("focus");
});
$("input").focus(function(){
$("<span>Focused!</span>").appendTo("body").fadeOut(1000);
});
```

serializeArray()

Encode a set of form elements as an array of names and values.

The `.serializeArray()` method creates a JavaScript array of objects, ready to be encoded as a JSON string. It operates on a jQuery object representing a set of form elements. The form elements can be of several types:

```
<form>
 <div><input type="text" name="a" value="1" id="a" /></div>
```

```
 <div><input type="text" name="b" value="2" id="b" /></div>
 <div><input type="hidden" name="c" value="3" id="c" /></div>
 <div>
  <textarea name="d" rows="8" cols="40">4</textarea>
 </div>
 <div><select name="e">
  <option value="5" selected="selected">5</option>
  <option value="6">6</option>
  <option value="7">7</option>
 </select></div>
 <div>
  <input type="checkbox" name="f" value="8" id="f" />
 </div>
 <div>
  <input type="submit" name="g" value="Submit" id="g" />
 </div>
</form>
```

The `.serializeArray()` method uses the standard W3C rules for successful controls to determine which elements it should include; in particular the element cannot be disabled and must contain a `name` attribute. No submit button value is serialized since the form was not submitted using a button. Data from file select elements is not serialized.

This method can act on a jQuery object that has selected individual form elements, such as `<input>`, `<textarea>`, and `<select>`. However, it is typically easier to select the `<form>` tag itself for serialization:

```
$('form').submit(function() {
 console.log($(this).serializeArray());
 return false;
});
```

This produces the following data structure (provided that the browser supports `console.log`):

```
[
 {
  name: "a",
  value: "1"
 },
 {
  name: "b",
  value: "2"
 },
 {
  name: "c",
  value: "3"
 },
 {
  name: "d",
  value: "4"
 },
 {
  name: "e",
  value: "5"
 }
]
```

Example

Get the values from a form, iterate through them, and append them to a results display.

```
function showValues() {
   var fields = $(":input").serializeArray();
   $("#results").empty();
   jQuery.each(fields, function(i, field){
    $("#results").append(field.value + " ");
```

```
 });
}

$(":checkbox, :radio").click(showValues);
$("select").change(showValues);
showValues();
```

stop([clearQueue], [jumpToEnd])

Stop the currently-running animation on the matched elements.

Arguments

clearQueue - A Boolean indicating whether to remove queued animation as well. Defaults to `false`.
jumpToEnd - A Boolean indicating whether to complete the current animation immediately. Defaults to `false`.

When `.stop()` is called on an element, the currently-running animation (if any) is immediately stopped. If, for instance, an element is being hidden with `.slideUp()` when `.stop()` is called, the element will now still be displayed, but will be a fraction of its previous height. Callback functions are not called.

If more than one animation method is called on the same element, the later animations are placed in the effects queue for the element. These animations will not begin until the first one completes. When `.stop()` is called, the next animation in the queue begins immediately. If the `clearQueue` parameter is provided with a value of `true`, then the rest of the animations in the queue are removed and never run.

If the `jumpToEnd` argument is provided with a value of `true`, the current animation stops, but the element is immediately given its target values for each CSS property. In our above `.slideUp()` example, the element would be immediately hidden. The callback function is then immediately called, if provided.

As of jQuery 1.7, if the first argument is provided as a string, only the animations in the queue represented by that string will be stopped.

The usefulness of the `.stop()` method is evident when we need to animate an element on `mouseenter` and `mouseleave`:

```
<div id="hoverme">
 Hover me
 <img id="hoverme" src="book.png" alt="" width="100" height="123" />
</div>
```

We can create a nice fade effect without the common problem of multiple queued animations by adding `.stop(true, true)` to the chain:

```
$('#hoverme-stop-2').hover(function() {
 $(this).find('img').stop(true, true).fadeOut();
}, function() {
 $(this).find('img').stop(true, true).fadeIn();
});
```

Toggling Animations

As of jQuery 1.7, stopping a toggled animation prematurely with `.stop()` will trigger jQuery's internal effects tracking. In previous versions, calling the `.stop()` method before a toggled animation was completed would cause the animation to lose track of its state (if jumpToEnd was false). Any subsequent animations would start at a new "half-way" state, sometimes resulting in the element disappearing. To observe the new behavior, see the final example below.

Animations may be stopped globally by setting the property `$.fx.off` to `true`. When this is done, all animation methods will immediately set elements to their final state when called, rather than displaying an effect.

Example

Click the Go button once to start the animation, then click the STOP button to stop it where it's currently positioned. Another option is to click several buttons to queue them up and see that stop just kills the currently playing one.

```
/* Start animation */
$("#go").click(function(){
$(".block").animate({left: '+=100px'}, 2000);
});
```

```
/* Stop animation when button is clicked */
$("#stop").click(function(){
$(".block").stop();
});

/* Start animation in the opposite direction */
$("#back").click(function(){
$(".block").animate({left: '-=100px'}, 2000);
});
```

Example

Click the slideToggle button to start the animation, then click again before the animation is completed. The animation will toggle the other direction from the saved starting point.

```
var $block = $('.block');
/* Toggle a sliding animation animation */
$('#toggle').on('click', function() {
  $block.stop().slideToggle(1000);
});
```

andSelf()

Add the previous set of elements on the stack to the current set.

As described in the discussion for `.end()`, jQuery objects maintain an internal stack that keeps track of changes to the matched set of elements. When one of the DOM traversal methods is called, the new set of elements is pushed onto the stack. If the previous set of elements is desired as well, `.andSelf()` can help.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

The result of the following code is a red background behind items 3, 4 and 5:

```
$('li.third-item').nextAll().andSelf()
 .css('background-color', 'red');
```

First, the initial selector locates item 3, initializing the stack with the set containing just this item. The call to `.nextAll()` then pushes the set of items 4 and 5 onto the stack. Finally, the `.andSelf()` invocation merges these two sets together, creating a jQuery object that points to all three items in document order: `{[<li.third-item>,<li>,<li> ]}`.

Example

Find all `div`s, and all the paragraphs inside of them, and give them both class names. Notice the `div` doesn't have the yellow background color since it didn't use `.andSelf()`.

```
$("div").find("p").andSelf().addClass("border");
  $("div").find("p").addClass("background");
```

prevAll([selector])

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.prevAll()` method searches through the predecessors of these elements in the DOM tree and construct a new jQuery object from the matching elements; the elements are returned in order beginning with the closest sibling.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the elements which come before it:

```
$('li.third-item').prevAll().css('background-color', 'red');
```

The result of this call is a red background behind items 1 and 2. Since we do not supply a selector expression, these preceding elements are unequivocally included as part of the object. If we had supplied one, the elements would be tested for a match before they were included.

Example
Locate all the divs preceding the last div and give them a class.

```
$("div:last").prevAll().addClass("before");
```

nextAll([selector])
Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.nextAll()` method allows us to search through the successors of these elements in the DOM tree and construct a new jQuery object from the matching elements.

The method optionally accepts a selector expression of the same type that we can pass to the `$()` function. If the selector is supplied, the elements will be filtered by testing whether they match it.

Consider a page with a simple list on it:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li class="third-item">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

If we begin at the third item, we can find the elements which come after it:

```
$('li.third-item').nextAll().css('background-color', 'red');
```

The result of this call is a red background behind items 4 and 5. Since we do not supply a selector expression, these following elements are

unequivocally included as part of the object. If we had supplied one, the elements would be tested for a match before they were included.

Example
Locate all the divs after the first and give them a class.

```
$("div:first").nextAll().addClass("after");
```

Example
Locate all the paragraphs after the second child in the body and give them a class.

```
$(":nth-child(1)").nextAll("p").addClass("after");
```

contents()
Get the children of each element in the set of matched elements, including text and comment nodes.

Given a jQuery object that represents a set of DOM elements, the `.contents()` method allows us to search through the immediate children of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.contents()` and `.children()` methods are similar, except that the former includes text nodes as well as HTML elements in the resulting jQuery object.

The `.contents()` method can also be used to get the content document of an iframe, if the iframe is on the same domain as the main page.

Consider a simple `<div>` with a number of text nodes, each of which is separated by two line break elements (`<br />`):

```
<div class="container">
 Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
 do eiusmod tempor incididunt ut labore et dolore magna aliqua.
 <br /><br />
 Ut enim ad minim veniam, quis nostrud exercitation ullamco
 laboris nisi ut aliquip ex ea commodo consequat.
 <br /> <br />
 Duis aute irure dolor in reprehenderit in voluptate velit
 esse cillum dolore eu fugiat nulla pariatur.
</div>
```

We can employ the `.contents()` method to help convert this blob of text into three well-formed paragraphs:

```
$('.container').contents().filter(function() {
 return this.nodeType == 3;
})
 .wrap('<p></p>')
.end()
.filter('br')
 .remove();
```

This code first retrieves the contents of `<div class="container">` and then filters it for text nodes, which are wrapped in paragraph tags. This is accomplished by testing the .nodeType property of the element. This DOM property holds a numeric code indicating the node's type; text nodes use the code 3. The contents are again filtered, this time for `<br />` elements, and these elements are removed.

Example
Find all the text nodes inside a paragraph and wrap them with a bold tag.

```
$("p").contents().filter(function(){ return this.nodeType != 1; }).wrap("<b/>");
```

Example
Change the background colour of links inside of an iframe.

```
$("#frameDemo").contents().find("a").css("background-color","#BADA55");
```

jQuery.param(obj)

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

Arguments

obj - An array or object to serialize.

This function is used internally to convert form element values into a serialized string representation (See .serialize() for more information).

As of jQuery 1.3, the return value of a function is used instead of the function as a String.

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. You can disable this functionality globally by setting `jQuery.ajaxSettings.traditional = true;`.

If the object passed is in an Array, it must be an array of objects in the format returned by .serializeArray()

```
[{name:"first",value:"Rick"},
{name:"last",value:"Astley"},
{name:"job",value:"Rock Star"}]
```

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

Note: Because there is no universally agreed-upon specification for param strings, it is not possible to encode complex data structures using this method in a manner that works ideally across all languages supporting such input. Until such time that there is, the `$.param` method will remain in its current form.

In jQuery 1.4, HTML5 input elements are also serialized.

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));

alert(recursiveEncoded);
alert(recursiveDecoded);
```

The values of `recursiveEncoded` and `recursiveDecoded` are alerted 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);
```

```
alert(shallowEncoded);
alert(shallowDecoded);
```

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

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

Example
Serialize a key/value object.

```
var params = { width:1680, height:1050 };
  var str = jQuery.param(params);
  $("#results").text(str);
```

Example
Serialize a few complex objects

```
// <=1.3.2:
$.param({ a: [2,3,4] }) // "a=2&a=3&a=4"
// >=1.4:
$.param({ a: [2,3,4] }) // "a[]=2&a[]=3&a[]=4"

// <=1.3.2:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a=[object+Object]&d=3&d=4&d=[object+Object]"
// >=1.4:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a[b]=1&a[c]=2&d[]=3&d[]=4&d[2][e]=5"
```

jQuery.isFunction(obj)
Determine if the argument passed is a Javascript function object.

Arguments
obj - Object to test whether or not it is a function.

Note: As of jQuery 1.3, functions provided by the browser like `alert()` and DOM element methods like `getAttribute()` are not guaranteed to be detected as functions in browsers such as Internet Explorer.

Example
Test a few parameter examples.

```
function stub() {
  }
  var objs = [
     function () {},
     { x:15, y:20 },
     null,
     stub,
     "function"
    ];

  jQuery.each(objs, function (i) {
   var isFunc = jQuery.isFunction(objs[i]);
   $("span").eq(i).text(isFunc);
  });
```

Example
Finds out if the parameter is a function.

```
$.isFunction(function(){});
```

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

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

Arguments
value - The value to search for.
array - An array through which to search.
fromIndex - The index of the array at which to begin the search. The default is 0, which will search the whole array.

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), 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.

Example
Report the index of some elements in the array.

```
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));
```

jQuery.makeArray(obj)
Convert an array-like object into a true JavaScript array.

Arguments
obj - Any object to turn into a native 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.

Example
Turn a collection of HTMLElements into an Array of them.

```
var elems = document.getElementsByTagName("div"); // returns a nodeList
  var arr = jQuery.makeArray(elems);
  arr.reverse(); // use an Array method on list of dom elements
  $(arr).appendTo(document.body);
```

Example
Turn a jQuery object into an array

```
var obj = $('li');
  var arr = $.makeArray(obj);
```

position()
Get the current coordinates of the first element in the set of matched elements, relative to the offset parent.

The `.position()` method allows us to retrieve the current position of an element *relative to the offset parent*. Contrast this with `.offset()`, which retrieves the current position *relative to the document*. When positioning a new element near another one and within the same containing DOM element, `.position()` is the more useful.

Returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the position coordinates of hidden elements or accounting for borders, margins, or padding set on the body

element.

Example
Access the position of the second paragraph:

```
var p = $("p:first");
var position = p.position();
$("p:last").text( "left: " + position.left + ", top: " + position.top );
```

offset()
Get the current coordinates of the first element in the set of matched elements, relative to the document.

The `.offset()` method allows us to retrieve the current position of an element *relative to the document*. Contrast this with `.position()`, which retrieves the current position *relative to the offset parent*. When positioning a new element on top of an existing one for global manipulation (in particular, for implementing drag-and-drop), `.offset()` is the more useful.

`.offset()` returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the offset coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.

While it is possible to get the coordinates of elements with `visibility:hidden` set, `display:none` is excluded from the rendering tree and thus has a position that is undefined.

Example
Access the offset of the second paragraph:

```
var p = $("p:last");
var offset = p.offset();
p.html( "left: " + offset.left + ", top: " + offset.top );
```

Example
Click to see the offset.

```
$("*", document.body).click(function (e) {
 var offset = $(this).offset();
 e.stopPropagation();
 $("#result").text(this.tagName + " coords ( " + offset.left + ", " +
                offset.top + " )");
});
```

offset(coordinates)
Set the current coordinates of every element in the set of matched elements, relative to the document.

Arguments
coordinates - An object containing the properties `top` and `left`, which are integers indicating the new top and left coordinates for the elements.

The `.offset()` setter method allows us to reposition an element. The element's position is specified *relative to the document*. If the element's `position` style property is currently `static`, it will be set to `relative` to allow for this repositioning.

Example
Set the offset of the second paragraph:

```
$("p:last").offset({ top: 10, left: 30 });
```

replaceAll(target)
Replace each target element with the set of matched elements.

Arguments

target - A selector expression indicating which element(s) to replace.

The `.replaceAll()` method is corollary to `.replaceWith()`, but with the source and target reversed. Consider this DOM structure:

```
<div class="container">
 <div class="inner first">Hello</div>
 <div class="inner second">And</div>
 <div class="inner third">Goodbye</div>
</div>
```

We can create an element, then replace other elements with it:

```
$('<h2>New heading</h2>').replaceAll('.inner');
```

This causes all of them to be replaced:

```
<div class="container">
 <h2>New heading</h2>
 <h2>New heading</h2>
 <h2>New heading</h2>
</div>
```

Or, we could select an element to use as the replacement:

```
$('.first').replaceAll('.third');
```

This results in the DOM structure:

```
<div class="container">
 <div class="inner second">And</div>
 <div class="inner first">Hello</div>
</div>
```

From this example, we can see that the selected element replaces the target by being moved from its old location, not by being cloned.

Example
Replace all the paragraphs with bold words.

```
$("<b>Paragraph. </b>").replaceAll("p"); // check replaceWith() examples
```

replaceWith(newContent)
Replace each element in the set of matched elements with the provided new content.

Arguments
newContent - The content to insert. May be an HTML string, DOM element, or jQuery object.

The `.replaceWith()` method removes content from the DOM and inserts new content in its place with a single call. Consider this DOM structure:

```
<div class="container">
 <div class="inner first">Hello</div>
 <div class="inner second">And</div>
 <div class="inner third">Goodbye</div>
</div>
```

The second inner `<div>` could be replaced with the specified HTML:

```
$('div.second').replaceWith('<h2>New heading</h2>');
```

This results in the structure:

```
<div class="container">
 <div class="inner first">Hello</div>
```

```
 <h2>New heading</h2>
 <div class="inner third">Goodbye</div>
</div>
```

All inner `<div>` elements could be targeted at once:

```
$('div.inner').replaceWith('<h2>New heading</h2>');
```

This causes all of them to be replaced:

```
<div class="container">
 <h2>New heading</h2>
 <h2>New heading</h2>
 <h2>New heading</h2>
</div>
```

An element could also be selected as the replacement:

```
$('div.third').replaceWith($('.first'));
```

This results in the DOM structure:

```
<div class="container">
 <div class="inner second">And</div>
 <div class="inner first">Hello</div>
</div>
```

This example demonstrates that the selected element replaces the target by being moved from its old location, not by being cloned.

The `.replaceWith()` method, like most jQuery methods, returns the jQuery object so that other methods can be chained onto it. However, it must be noted that the *original* jQuery object is returned. This object refers to the element that has been removed from the DOM, not the new element that has replaced it.

As of jQuery 1.4, `.replaceWith()` can also work on disconnected DOM nodes. For example, with the following code, `.replaceWith()` returns a jQuery set containing only a paragraph.:

```
$("<div/>").replaceWith("<p></p>");
```

The `.replaceWith()` method can also take a function as its argument:

```
$('div.container').replaceWith(function() {
 return $(this).contents();
});
```

This results in `<div class="container">` being replaced by its three child `<div>`s. The return value of the function may be an HTML string, DOM element, or jQuery object.

Example

On click, replace the button with a div containing the same word.

```
$("button").click(function () {
 $(this).replaceWith( "<div>" + $(this).text() + "</div>" );
});
```

Example

Replace all paragraphs with bold words.

```
$("p").replaceWith( "<b>Paragraph. </b>" );
```

Example

On click, replace each paragraph with a div that is already in the DOM and selected with the `$()` function. Notice it doesn't clone the object but rather moves it to replace the paragraph.

```
$("p").click(function () {
 $(this).replaceWith( $("div") );
});
```

Example
On button click, replace the containing div with its child divs and append the class name of the selected element to the paragraph.

```
$('button').bind("click", function() {
 var $container = $("div.container").replaceWith(function() {
  return $(this).contents();
 });

 $("p").append( $container.attr("class") );
});
```

wrapInner(wrappingElement)
Wrap an HTML structure around the content of each element in the set of matched elements.

Arguments
wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the content of the matched elements.

The `.wrapInner()` function can take any string or object that could be passed to the `$()` factory function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. The structure will be wrapped around the content of each of the elements in the set of matched elements.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using `.wrapInner()`, we can insert an HTML structure around the content of each inner `<div>` elements like so:

```
$('.inner').wrapInner('<div class="new" />');
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around the content of each matched element:

```
<div class="container">
 <div class="inner">
  <div class="new">Hello</div>
 </div>
 <div class="inner">
  <div class="new">Goodbye</div>
 </div>
</div>
```

The second version of this method allows us to instead specify a callback function. This callback function will be called once for every matched element; it should return a DOM element, jQuery object, or HTML snippet in which to wrap the content of the corresponding element. For example:

```
$('.inner').wrapInner(function() {
 return '<div class="' + this.nodeValue + '" />';
});
```

This will cause each `<div>` to have a class corresponding to the text it wraps:

```
<div class="container">
 <div class="inner">
  <div class="Hello">Hello</div>
 </div>
```

```
 <div class="inner">
  <div class="Goodbye">Goodbye</div>
 </div>
</div>
```

Note: When passing a selector string to the `.wrapInner()` function, the expected input is well formed HTML with correctly closed tags. Examples of valid input include:

```
$(elem).wrapInner("<div class='test' />");
$(elem).wrapInner("<div class='test'></div>");
$(elem).wrapInner("<div class="test"></div>");
```

Example
Selects all paragraphs and wraps a bold tag around each of its contents.

```
$("p").wrapInner("<b></b>");
```

Example
Wraps a newly created tree of objects around the inside of the body.

```
$("body").wrapInner("<div><div><p><em><b></b></em></p></div></div>");
```

Example
Selects all paragraphs and wraps a bold tag around each of its contents.

```
$("p").wrapInner(document.createElement("b"));
```

Example
Selects all paragraphs and wraps a jQuery object around each of its contents.

```
$("p").wrapInner($("<span class='red'></span>"));
```

wrapAll(wrappingElement)
Wrap an HTML structure around all elements in the set of matched elements.

Arguments
wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the matched elements.

The `.wrapAll()` function can take any string or object that could be passed to the `$()` function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. The structure will be wrapped around all of the elements in the set of matched elements, as a single group.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using `.wrapAll()`, we can insert an HTML structure around the inner `<div>` elements like so:

```
$('.inner').wrapAll('<div class="new" />');
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around all matched elements:

```
<div class="container">
 <div class="new">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
```

```
 </div>
</div>
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrapAll("<div></div>");
```

Example
Wraps a newly created tree of objects around the spans. Notice anything in between the spans gets left out like the (red text) in this example. Even the white space between spans is left out. Click View Source to see the original html.

```
$("span").wrapAll("<div><div><p><em><b></b></em></p></div></div>");
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrapAll(document.createElement("div"));
```

Example
Wrap a jQuery object double depth div around all of the paragraphs. Notice it doesn't move the object but just clones it to wrap around its target.

```
$("p").wrapAll($(".doublediv"));
```

map(callback(index, domElement))
Pass each element in the current matched set through a function, producing a new jQuery object containing the return values.

Arguments
callback(index, domElement) - A function object that will be invoked for each element in the current set.

As the return value is a jQuery-wrapped array, it's very common to `get()` the returned object to work with a basic array.

The `.map()` method is particularly useful for getting or setting the value of a collection of elements. Consider a form with a set of checkboxes in it:

```
<form method="post" action="">
 <fieldset>
  <div>
   <label for="two">2</label>
   <input type="checkbox" value="2" id="two" name="number[]">
  </div>
  <div>
   <label for="four">4</label>
   <input type="checkbox" value="4" id="four" name="number[]">
  </div>
  <div>
   <label for="six">6</label>
   <input type="checkbox" value="6" id="six" name="number[]">
  </div>
  <div>
   <label for="eight">8</label>
   <input type="checkbox" value="8" id="eight" name="number[]">
  </div>
 </fieldset>
</form>
```

We can get a comma-separated list of checkbox `ID`s:

```
$(':checkbox').map(function() {
 return this.id;
}).get().join(',');
```

The result of this call is the string, "two,four,six,eight".

Within the callback function, this refers to the current DOM element for each iteration. The function can return an individual data item or an array of data items to be inserted into the resulting set. If an array is returned, the elements inside the array are inserted into the set. If the function returns null or undefined, no element will be inserted.

Example
Build a list of all the values within a form.

```
$("p").append( $("input").map(function(){
   return $(this).val();
  }).get().join(", ") );
```

Example
A contrived example to show some functionality.

```
var mappedItems = $("li").map(function (index) {
 var replacement = $("<li>").text($(this).text()).get(0);
 if (index == 0) {
  /* make the first item all caps */
  $(replacement).text($(replacement).text().toUpperCase());
 } else if (index == 1 || index == 3) {
  /* delete the second and fourth items */
  replacement = null;
 } else if (index == 2) {
  /* make two of the third item and add some text */
  replacement = [replacement,$("<li>").get(0)];
  $(replacement[0]).append("<b> - A</b>");
  $(replacement[1]).append("Extra <b> - B</b>");
 }

 /* replacement will be a dom element, null,
   or an array of dom elements */
 return replacement;
});
$("#results").append(mappedItems);
```

Example
Equalize the heights of the divs.

```
$.fn.equalizeHeights = function() {
 var maxHeight = this.map(function(i,e) {
  return $(e).height();
 }).get();

 return this.height( Math.max.apply(this, maxHeight) );
};

$('input').click(function(){
 $('div').equalizeHeights();
});
```

hasClass(className)
Determine whether any of the matched elements are assigned the given class.

Arguments
className - The class name to search for.

Elements may have more than one class assigned to them. In HTML, this is represented by separating the class names with a space:

```
<div id="mydiv" class="foo bar"></div>
```

The `.hasClass()` method will return `true` if the class is assigned to an element, even if other classes also are. For example, given the HTML above, the following will return `true`:

```
$('#mydiv').hasClass('foo')
```

As would:

```
$('#mydiv').hasClass('bar')
```

While this would return `false`:

```
$('#mydiv').hasClass('quux')
```

Example
Looks for the paragraph that contains 'selected' as a class.

```
$("div#result1").append($("p:first").hasClass("selected").toString());
$("div#result2").append($("p:last").hasClass("selected").toString());
$("div#result3").append($("p").hasClass("selected").toString());
```

Version 1.2.3

jQuery.removeData(element, [name])
Remove a previously-stored piece of data.

Arguments
element - A DOM element from which to remove data.
name - A string naming the piece of data to remove.

Note: This is a low-level method, you should probably use `.removeData()` instead.

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

Example
Set a data store for 2 names then remove one of them.

```
var div = $("div")[0];
$("span:eq(0)").text("" + $("div").data("test1"));
jQuery.data(div, "test1", "VALUE-1");
jQuery.data(div, "test2", "VALUE-2");
$("span:eq(1)").text("" + jQuery.data(div, "test1"));
jQuery.removeData(div, "test1");
$("span:eq(2)").text("" + jQuery.data(div, "test1"));
$("span:eq(3)").text("" + jQuery.data(div, "test2"));
```

jQuery.data(element, key, value)
Store arbitrary data associated with the specified element. Returns the value that was set.

Arguments
element - The DOM element to associate with the data.
key - A string naming the piece of data to set.
value - The new data value.

Note: This is a low-level method; a more convenient `.data()` is also available.

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');
```

Note: 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.

Example
Store then retrieve a value from the div element.

```
var div = $("div")[0];
  jQuery.data(div, "test", { first: 16, last: "pizza!" });
  $("span:first").text(jQuery.data(div, "test").first);
  $("span:last").text(jQuery.data(div, "test").last);
```

jQuery.data(element, key)
Returns value at named data store for the element, as set by `jQuery.data(element, name, value)`, or the full data store for the element.

Arguments
element - The DOM element to query for the data.
key - Name of the data stored.

Note: This is a low-level method; a more convenient `.data()` is also available.

Regarding HTML5 data-* attributes: This low-level method does NOT retrieve the `data-*` attributes unless the more convenient `.data()` method has already retrieved them.

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 from memory leaks. We can retrieve several distinct values for a single element one at a time, or as a set:

```
alert(jQuery.data( document.body, 'foo' ));
alert(jQuery.data( document.body ));
```

The above lines alert the data values that were set on the `body` element. If nothing was set on that element, an empty string is returned.

Calling `jQuery.data(element)` retrieves all of the element's associated values as a JavaScript object. Note that jQuery itself uses this method to store data for internal use, such as event handlers, so do not assume that it contains only data that your own code has stored.

Note: 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.

Example
Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value, div = $("div")[0];

 switch ($("button").index(this)) {
  case 0 :
   value = jQuery.data(div, "blah");
   break;
  case 1 :
   jQuery.data(div, "blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   jQuery.data(div, "blah", 86);
   value = "Stored!";
   break;
  case 3 :
   jQuery.removeData(div, "blah");
```

```
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

removeData([name])

Remove a previously-stored piece of data.

Arguments

name - A string naming the piece of data to delete.

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. Removing data from jQuery's internal `.data()` cache does not effect any HTML5 `data-` attributes in a document; use `.removeAttr()` to remove those.

When using `.removeData("name")`, jQuery will attempt to locate a `data-` attribute on the element if no property by that name is in the internal data cache. To avoid a re-query of the `data-` attribute, set the name to a value of either `null` or `undefined` (e.g. `.data("name", undefined)`) rather than using `.removeData()`.

As of jQuery 1.7, when called with an array of keys or a string of space-separated keys, `.removeData()` deletes the value of each key in that array or string.

As of jQuery 1.4.3, calling `.removeData()` will cause the value of the property being removed to revert to the value of the data attribute of the same name in the DOM, rather than being set to `undefined`.

Example

Set a data store for 2 names then remove one of them.

```
$("span:eq(0)").text("" + $("div").data("test1"));
  $("div").data("test1", "VALUE-1");
  $("div").data("test2", "VALUE-2");
  $("span:eq(1)").text("" + $("div").data("test1"));
  $("div").removeData("test1");
  $("span:eq(2)").text("" + $("div").data("test1"));
  $("span:eq(3)").text("" + $("div").data("test2"));
```

data(key, value)

Store arbitrary data associated with the matched elements.

Arguments

key - A string naming the piece of data to set.
value - The new data value; it can be any Javascript type including Array or Object.

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 later:

```
$('body').data('foo', 52);
$('body').data('bar', { myType: 'test', count: 40 });

$('body').data('foo'); // 52
$('body').data(); // {foo: 52, bar: { myType: 'test', count: 40 }}
```

In jQuery 1.4.3 setting an element's data object with `.data(obj)` extends the data previously stored with that element. jQuery itself uses the `.data()` method to save information under the names 'events' and 'handle', and also reserves any data name starting with an underscore ('_') for internal use.

Prior to jQuery 1.4.3 (starting in jQuery 1.4) the .data() method completely replaced all data, instead of just extending the data object. If you are using third-party plugins it may not be advisable to completely replace the element's data object, since plugins may have also set data.

Due to the way browsers interact with plugins and external code, the `.data()` method cannot be used on `<object>` (unless it's a Flash plugin), `<applet>` or `<embed>` elements.

Example
Store then retrieve a value from the div element.

```
$("div").data("test", { first: 16, last: "pizza!" });
$("span:first").text($("div").data("test").first);
$("span:last").text($("div").data("test").last);
```

data(key)
Returns value at named data store for the first element in the jQuery collection, as set by data(name, value).

Arguments
key - Name of the data stored.

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 retrieve several distinct values for a single element one at a time, or as a set:

```
alert($('body').data('foo'));
alert($('body').data());
```

The above lines alert the data values that were set on the `body` element. If no data at all was set on that element, `undefined` is returned.

```
alert( $("body").data("foo")); //undefined
$("body").data("bar", "foobar");
alert( $("body").data("bar")); //foobar
```

HTML5 data-* Attributes

As of jQuery 1.4.3 HTML 5 data- attributes will be automatically pulled in to jQuery's data object. The treatment of attributes with embedded dashes was changed in jQuery 1.6 to conform to the W3C HTML5 specification.

For example, given the following HTML:

```
<div data-role="page" data-last-value="43" data-hidden="true" data-options='{"name":"John"}'></div>
```

All of the following jQuery code will work.

```
$("div").data("role") === "page";
$("div").data("lastValue") === 43;
$("div").data("hidden") === true;
$("div").data("options").name === "John";
```

Every attempt is made to convert the string to a JavaScript value (this includes booleans, numbers, objects, arrays, and null) otherwise it is left as a string. To retrieve the value's attribute as a string without any attempt to convert it, use the `attr()` method. When the data attribute is an object (starts with '{') or array (starts with '[') then `jQuery.parseJSON` is used to parse the string; it must follow valid JSON syntax*including quoted property names*. The data- attributes are pulled in the first time the data property is accessed and then are no longer accessed or mutated (all data values are then stored internally in jQuery).

Calling `.data()` with no parameters retrieves all of the values as a JavaScript object. This object can be safely cached in a variable as long as a new object is not set with `.data(obj)`. Using the object directly to get or set values is faster than making individual calls to `.data()` to get or set each value:

```
var mydata = $("#mydiv").data();
if ( mydata.count < 9 ) {
  mydata.count = 43;
  mydata.status = "embiggened";
}
```

Example
Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value;

 switch ($("button").index(this)) {
  case 0 :
   value = $("div").data("blah");
   break;
  case 1 :
   $("div").data("blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   $("div").data("blah", 86);
   value = "Stored!";
   break;
  case 3 :
   $("div").removeData("blah");
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

Version 1.2.6

event.timeStamp
The difference in milliseconds between the time the browser created the event and January 1, 1970.

This property can be useful for profiling event performance by getting the `event.timeStamp` value at two points in the code and noting the difference. To simply determine the current time inside an event handler, use `(new Date).getTime()` instead.

Note: Due to a bug open since 2004, this value is not populated correctly in Firefox and it is not possible to know the time the event was created in that browser.

Example
Display the time since the click handler last executed.

```
var last, diff;
$('div').click(function(event) {
 if ( last ) {
  diff = event.timeStamp - last
  $('div').append('time since last event: ' + diff + '<br/>');
 } else {
  $('div').append('Click again.<br/>');
 }
 last = event.timeStamp;
});
```

outerWidth([includeMargin])

Get the current computed width for the first element in the set of matched elements, including padding and border.

Arguments

includeMargin - A Boolean indicating whether to include the element's margin in the calculation.

Returns the width of the element, along with left and right padding, border, and optionally margin, in pixels.

If `includeMargin` is omitted or `false`, the padding and border are included in the calculation; if `true`, the margin is also included.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.

Example

Get the outerWidth of a paragraph.

```
var p = $("p:first");
$("p:last").text( "outerWidth:" + p.outerWidth()+ " , outerWidth(true):" + p.outerWidth(true) );
```

outerHeight([includeMargin])

Get the current computed height for the first element in the set of matched elements, including padding, border, and optionally margin. Returns an integer (without "px") representation of the value or null if called on an empty set of elements.

Arguments

includeMargin - A Boolean indicating whether to include the element's margin in the calculation.

The top and bottom padding and border are always included in the `.outerHeight()` calculation; if the `includeMargin` argument is set to `true`, the margin (top and bottom) is also included.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.

Example

Get the outerHeight of a paragraph.

```
var p = $("p:first");
$("p:last").text( "outerHeight:" + p.outerHeight() + " , outerHeight(true):" + p.outerHeight(true) );
```

innerWidth()

Get the current computed width for the first element in the set of matched elements, including padding but not border.

This method returns the width of the element, including left and right padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.width()` instead.

Example

Get the innerWidth of a paragraph.

```
var p = $("p:first");
$("p:last").text( "innerWidth:" + p.innerWidth() );
```

innerHeight()

Get the current computed height for the first element in the set of matched elements, including padding but not border.

This method returns the height of the element, including top and bottom padding, in pixels.

This method is not applicable to `window` and `document` objects; for these, use `.height()` instead.

Example
Get the innerHeight of a paragraph.

```
var p = $("p:first");
$("p:last").text( "innerHeight:" + p.innerHeight() );
```

scrollLeft()
Get the current horizontal position of the scroll bar for the first element in the set of matched elements.

The horizontal scroll position is the same as the number of pixels that are hidden from view to the left of the scrollable area. If the scroll bar is at the very left, or if the element is not scrollable, this number will be `0`.

Note: `.scrollLeft()`, when called directly or animated as a property using `.animate()`, will not work if the element it is being applied to is hidden.

Example
Get the scrollLeft of a paragraph.

```
var p = $("p:first");
$("p:last").text( "scrollLeft:" + p.scrollLeft() );
```

scrollLeft(value)
Set the current horizontal position of the scroll bar for each of the set of matched elements.

Arguments
value - An integer indicating the new position to set the scroll bar to.

The horizontal scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollLeft` positions the horizontal scroll of each matched element.

Example
Set the scrollLeft of a div.

```
$("div.demo").scrollLeft(300);
```

scrollTop()
Get the current vertical position of the scroll bar for the first element in the set of matched elements.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. If the scroll bar is at the very top, or if the element is not scrollable, this number will be `0`.

Example
Get the scrollTop of a paragraph.

```
var p = $("p:first");
$("p:last").text( "scrollTop:" + p.scrollTop() );
```

scrollTop(value)
Set the current vertical position of the scroll bar for each of the set of matched elements.

Arguments
value - An integer indicating the new position to set the scroll bar to.

The vertical scroll position is the same as the number of pixels that are hidden from view above the scrollable area. Setting the `scrollTop` positions the vertical scroll of each matched element.

Example
Set the scrollTop of a div.

```
$("div.demo").scrollTop(300);
```

Version 1.3

event.isImmediatePropagationStopped()
Returns whether event.stopImmediatePropagation() was ever called on this event object.

This property was introduced in DOM level 3.

Example
Checks whether event.stopImmediatePropagation() was called.

```
function immediatePropStopped(e) {
 var msg = "";
 if ( e.isImmediatePropagationStopped() ) {
  msg = "called"
 } else {
  msg = "not called";
 }
 $("#stop-log").append( "<div>" + msg + "</div>" );
}

$("button").click(function(event) {
 immediatePropStopped(event);
 event.stopImmediatePropagation();
 immediatePropStopped(event);
});
```

event.stopImmediatePropagation()
Keeps the rest of the handlers from being executed and prevents the event from bubbling up the DOM tree.

In addition to keeping any additional handlers on an element from being executed, this method also stops the bubbling by implicitly calling `event.stopPropagation()`. To simply prevent the event from bubbling to ancestor elements but allow other event handlers to execute on the same element, we can use `event.stopPropagation()` instead.

Use `event.isImmediatePropagationStopped()` to know whether this method was ever called (on that event object).

Example
Prevents other event handlers from being called.

```
$("p").click(function(event){
 event.stopImmediatePropagation();
});
$("p").click(function(event){
 // This function won't be executed
 $(this).css("background-color", "#f00");
});
$("div").click(function(event) {
 // This function will be executed
  $(this).css("background-color", "#f00");
});
```

event.isPropagationStopped()

Returns whether event.stopPropagation() was ever called on this event object.

This event method is described in the W3C DOM Level 3 specification.

Example

Checks whether event.stopPropagation() was called

```
function propStopped(e) {
 var msg = "";
 if ( e.isPropagationStopped() ) {
  msg = "called"
 } else {
  msg = "not called";
 }
 $("#stop-log").append( "<div>" + msg + "</div>" );
}

$("button").click(function(event) {
 propStopped(event);
 event.stopPropagation();
 propStopped(event);
});
```

event.isDefaultPrevented()

Returns whether event.preventDefault() was ever called on this event object.

Example

Checks whether event.preventDefault() was called.

```
$("a").click(function(event){
 alert( event.isDefaultPrevented() ); // false
 event.preventDefault();
 alert( event.isDefaultPrevented() ); // true
});
```

event.result

The last value returned by an event handler that was triggered by this event, unless the value was `undefined`.

This property can be useful for getting previous return values of custom events.

Example

Display previous handler's return value

```
$("button").click(function(event) {
 return "hey";
});
$("button").click(function(event) {
 $("p").html( event.result );
});
```

event.currentTarget

The current DOM element within the event bubbling phase.

This property will typically be equal to the `this` of the function.

If you are using jQuery.proxy or another form of scope manipulation, `this` will be equal to whatever context you have provided, not `event.currentTarget`

Example
Alert that currentTarget matches the `this` keyword.

```
$("p").click(function(event) {
 alert( event.currentTarget === this ); // true
});
```

jQuery.fx.off
Globally disable all animations.

When this property is set to `true`, all animation methods will immediately set elements to their final state when called, rather than displaying an effect. This may be desirable for a couple reasons:

- jQuery is being used on a low-resource device.
- Users are encountering accessibility problems with the animations (see the article Turn Off Animation for more information).

Animations can be turned back on by setting the property to `false`.

Example
Toggle animation on and off

```
var toggleFx = function() {
 $.fx.off = !$.fx.off;
};
toggleFx();

$("button").click(toggleFx)

$("input").click(function(){
 $("div").toggle("slow");
});
```

pushStack(elements)
Add a collection of DOM elements onto the jQuery stack.

Arguments
elements - An array of elements to push onto the stack and make into a new jQuery object.

Example
Add some elements onto the jQuery stack, then pop back off again.

```
jQuery([])
  .pushStack( document.getElementsByTagName("div") )
    .remove()
  .end();
```

jQuery.dequeue(element, [queueName])
Execute the next function on the queue for the matched element.

Arguments
element - A DOM element from which to remove and execute a queued function.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Note: This is a low-level method, you should probably use `.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.

Example
Use jQuery.dequeue() to end a custom queue function which allows the queue to keep going.

```
$("button").click(function () {
   $("div").animate({left:'+=200px'}, 2000);
   $("div").animate({top:'0px'}, 600);
   $("div").queue(function () {
    $(this).toggleClass("red");
     $.dequeue( this );
       });
   $("div").animate({left:'10px', top:'30px'}, 700);
  });
```

jQuery.queue(element, [queueName])
Show the queue of functions to be executed on the matched element.

Arguments
element - A DOM element to inspect for an attached queue.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Note: This is a low-level method, you should probably use `.queue()` instead.

Example
Show the length of the queue.

```
$("#show").click(function () {
   var n = jQuery.queue( $("div")[0], "fx" );
   $("span").text("Queue length is: " + n.length);
  });
  function runIt() {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   $("div").slideToggle(1000);
   $("div").slideToggle("fast");
   $("div").animate({left:'-=200'},1500);
   $("div").hide("slow");
   $("div").show(1200);
   $("div").slideUp("normal", runIt);
  }
  runIt();
```

jQuery.queue(element, queueName, newQueue)
Manipulate the queue of functions to be executed on the matched element.

Arguments
element - A DOM element where the array of queued functions is attached.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.
newQueue - An array of functions to replace the current queue contents.

Note: This is a low-level method, you should probably use `.queue()` instead.

Every element can have one or more queues of functions attached to it by jQuery. In most applications, only one queue (called `fx`) is used. Queues allow a sequence of actions to be called on an element asynchronously, without halting program execution.

The `jQuery.queue()` method allows us to directly manipulate this queue of functions. Calling `jQuery.queue()` with a callback is particularly useful; it allows us to place a new function at the end of the queue.

Note that when adding a function with `jQuery.queue()`, we should ensure that `jQuery.dequeue()` is eventually called so that the next function in line executes.

Example

Queue a custom function.

```
$(document.body).click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},2000);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).addClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").animate({left:'-=200'},500);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).removeClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").slideUp();
  });
```

Example

Set a queue array to delete the queue.

```
$("#start").click(function () {
   $("div").show("slow");
   $("div").animate({left:'+=200'},5000);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).addClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").animate({left:'-=200'},1500);
   jQuery.queue( $("div")[0], "fx", function () {
    $(this).removeClass("newcolor");
    jQuery.dequeue( this );
   });
   $("div").slideUp();
  });
  $("#stop").click(function () {
   jQuery.queue( $("div")[0], "fx", [] );
   $("div").stop();
  });
```

die()

Remove all event handlers previously attached using `.live()` from the elements.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to calling `.unbind()` with no arguments, which is used to remove all handlers attached with `.bind()`.See the discussions of `.live()` and `.unbind()` for further details.

As of jQuery 1.7, use of `.die()` (and its complementary method, `.live()`) is not recommended. Instead, use .off() to remove event handlers bound with .on()

Note: In order for .die() to function correctly, the selector used with it must match exactly the selector initially used with .live().

die(eventType, [handler])

Remove an event handler previously attached using `.live()` from the elements.

Arguments

eventType - A string containing a JavaScript event type, such as `click` or `keydown`.
handler - The function that is no longer to be executed.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to `.unbind()`, which is used to remove handlers attached with `.bind()`.See the discussions of `.live()` and `.unbind()` for further details.

Note: In order for `.die()` to function correctly, the selector used with it must match exactly the selector initially used with `.live()`.

Example
Can bind and unbind events to the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("#theone").live("click", aClick)
       .text("Can Click!");
});
$("#unbind").click(function () {
 $("#theone").die("click", aClick)
       .text("Does nothing...");
});
```

Example
To unbind all live events from all paragraphs, write:

```
$("p").die()
```

Example
To unbind all live click events from all paragraphs, write:

```
$("p").die( "click" )
```

Example
To unbind just one previously bound handler, pass the function in as the second argument:

```
var foo = function () {
// code to handle some kind of event
};

$("p").live("click", foo); // ... now foo will be called when paragraphs are clicked ...

$("p").die("click", foo); // ... foo will no longer be called.
```

live(events, handler(eventObject))
Attach an event handler for all elements which match the current selector, now and in the future.

Arguments
events - A string containing a JavaScript event type, such as "click" or "keydown." As of jQuery 1.4 the string can contain multiple, space-separated event types or custom event names.
handler(eventObject) - A function to execute at the time the event is triggered.

As of jQuery 1.7, the `.live()` method is deprecated. Use .on() to attach event handlers. Users of older versions of jQuery should use .delegate() in preference to `.live()`.

This method provides a means to attach delegated event handlers to the `document` element of a page, which simplifies the use of event handlers when content is dynamically added to a page. See the discussion of direct versus delegated events in the .on() method for more information.

Rewriting the `.live()` method in terms of its successors is straightforward; these are templates for equivalent calls for all three event attachment methods:

```
$(selector).live(events, data, handler);        // jQuery 1.3+
$(document).delegate(selector, events, data, handler); // jQuery 1.4.3+
$(document).on(events, selector, data, handler);    // jQuery 1.7+
```

The `events` argument can either be a space-separated list of event type names and optional namespaces, or an `event-map` of event names strings

and handlers. The `data` argument is optional and can be omitted. For example, the following three method calls are functionally equivalent (but see below for more effective and performant ways to attach delegated event handlers):

```
$("a.offsite").live("click", function(){ alert("Goodbye!"); });        // jQuery 1.3+
$(document).delegate("a.offsite", "click", function(){ alert("Goodbye!"); }); // jQuery 1.4.3+
$(document).on("click", "a.offsite", function(){ alert("Goodbye!"); });    // jQuery 1.7+
```

Use of the `.live()` method is no longer recommended since later versions of jQuery offer better methods that do not have its drawbacks. In particular, the following issues arise with the use of `.live()`:

• jQuery attempts to retrieve the elements specified by the selector before calling the `.live()` method, which may be time-consuming on large documents.

• Chaining methods is not supported. For example, `$("a").find(".offsite, .external").live( ... );` is *not* valid and does not work as expected.

• Since all `.live()` events are attached at the `document` element, events take the longest and slowest possible path before they are handled.

• On mobile iOS (iPhone, iPad and iPod Touch) the `click` event does not bubble to the document body for most elements and cannot be used with `.live()` without applying one of the following workarounds:

• Use natively clickable elements such as a or `button`, as both of these do bubble to `document`.

• Use `.on()` or `.delegate()` attached to an element below the level of `document.body`, since mobile iOS does bubble within the body.

• Apply the CSS style `cursor:pointer` to the element that needs to bubble clicks (or a parent including `document.documentElement`). Note however, this will disable copypaste on the element and cause it to be highlighted when touched.

• Calling event.stopPropagation() in the event handler is ineffective in stopping event handlers attached lower in the document; the event has already propagated to `document`.

• The `.live()` method interacts with other event methods in ways that can be surprising, e.g., `$(document).unbind("click")` removes all click handlers attached by any call to `.live()`!

For pages still using `.live()`, this list of version-specific differences may be helpful:

• Before jQuery 1.7, to stop further handlers from executing after one bound using `.live()`, the handler must return `false`. Calling `.stopPropagation()` will not accomplish this.

• As of **jQuery 1.4** the `.live()` method supports custom events as well as *all JavaScript events that bubble*. It also supports certain events that don't bubble, including `change`, `submit`, `focus` and `blur`.

• In **jQuery 1.3.x** only the following JavaScript events could be bound: `click`, `dblclick`, `keydown`, `keypress`, `keyup`, `mousedown`, `mousemove`, `mouseout`, `mouseover`, and `mouseup`.

Example

Click a paragraph to add another. Note that .live() binds the click event to all paragraphs - even new ones.

```
$("p").live("click", function(){
 $(this).after("<p>Another paragraph!</p>");
});
```

Example

Cancel a default action and prevent it from bubbling up by returning false.

```
$("a").live("click", function() { return false; })
```

Example

Cancel only the default action by using the preventDefault method.

```
$("a").live("click", function(event){
 event.preventDefault();
});
```

Example

Bind custom events with .live().

```
$("p").live("myCustomEvent", function(e, myName, myValue) {
 $(this).text("Hi there!");
 $("span").stop().css("opacity", 1)
      .text("myName = " + myName)
      .fadeIn(30).fadeOut(1000);
});
```

```
$("button").click(function () {
 $("p").trigger("myCustomEvent");
});
```

Example

Use a map to bind multiple live event handlers. Note that .live() calls the click, mouseover, and mouseout event handlers for all paragraphs--even new ones.

```
$("p").live({
 click: function() {
  $(this).after("<p>Another paragraph!</p>");
 },
 mouseover: function() {
  $(this).addClass("over");
 },
 mouseout: function() {
  $(this).removeClass("over");
 }
});
```

closest(selector)

Get the first element that matches the selector, beginning at the current element and progressing up through the DOM tree.

Arguments

selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.closest()` method searches through these elements and their ancestors in the DOM tree and constructs a new jQuery object from the matching elements. The `.parents()` and `.closest()` methods are similar in that they both traverse up the DOM tree. The differences between the two, though subtle, are significant:

.closest()	.parents()
Begins with the current element	Begins with the parent element
Travels up the DOM tree until it finds a match for the supplied selector	Travels up the DOM tree to the document's root element, adding each ancestor element to a temporary collectio
The returned jQuery object contains zero or one element	The returned jQuery object contains zero, one, or multiple elements

```
<ul id="one" class="level-1">
 <li class="item-i">I</li>
 <li id="ii" class="item-ii">II
 <ul class="level-2">
  <li class="item-a">A</li>
  <li class="item-b">B
   <ul class="level-3">
    <li class="item-1">1</li>
    <li class="item-2">2</li>
    <li class="item-3">3</li>
   </ul>
  </li>
  <li class="item-c">C</li>
 </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

Suppose we perform a search for `<ul>` elements starting at item A:

```
$('li.item-a').closest('ul')
```

```
 .css('background-color', 'red');
```

This will change the color of the level-2 `<ul>`, since it is the first encountered when traveling up the DOM tree.

Suppose we search for an `<li>` element instead:

```
$('li.item-a').closest('li')
 .css('background-color', 'red');
```

This will change the color of list item A. The `.closest()` method begins its search *with the element itself* before progressing up the DOM tree, and stops when item A matches the selector.

We can pass in a DOM element as the context within which to search for the closest element.

```
var listItemII = document.getElementById('ii');
$('li.item-a').closest('ul', listItemII)
 .css('background-color', 'red');
$('li.item-a').closest('#one', listItemII)
 .css('background-color', 'green');
```

This will change the color of the level-2 `<ul>`, because it is both the first `<ul>` ancestor of list item A and a descendant of list item II. It will not change the color of the level-1 `<ul>`, however, because it is not a descendant of list item II.

Example
Show how event delegation can be done with closest. The closest list element toggles a yellow background when it or its descendent is clicked.

```
$( document ).bind("click", function( e ) {
  $( e.target ).closest("li").toggleClass("hilight");
 });
```

Example
Pass a jQuery object to closest. The closest list element toggles a yellow background when it or its descendent is clicked.

```
var $listElements = $("li").css("color", "blue");
 $( document ).bind("click", function( e ) {
  $( e.target ).closest( $listElements ).toggleClass("hilight");
 });
```

closest(selectors, [context])
Gets an array of all the elements and selectors matched against the current element up through the DOM tree.

Arguments
selectors - An array or string containing a selector expression to match elements against (can also be a jQuery object).
context - A DOM element within which a matching element may be found. If no context is passed in then the context of the jQuery set will be used instead.

This signature (only!) is deprecated as of jQuery 1.7. This method is primarily meant to be used internally or by plugin authors.

Example
Show how event delegation can be done with closest.

```
var close = $("li:first").closest(["ul", "body"]);
 $.each(close, function(i){
 $("li").eq(i).html( this.selector + ": " + this.elem.nodeName );
 });
```

context
The DOM node context originally passed to `jQuery()`; if none was passed then context will likely be the document.

The `.live()` method for binding event handlers uses this property to determine the root element to use for its event delegation needs.

The value of this property is typically equal to `document`, as this is the default context for jQuery objects if none is supplied. The context may differ if, for example, the object was created by searching within an `<iframe>` or XML document.

Note that the context property may only apply to the elements originally selected by `jQuery()`, as it is possible for the user to add elements to the collection via methods such as `.add()` and these may have a different context.

Example
Determine the exact context used.

```
$("ul")
 .append("<li>" + $("ul").context + "</li>")
 .append("<li>" + $("ul", document.body).context.nodeName + "</li>");
```

toggle([duration], [callback])
Display or hide the matched elements.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

Note: The event handling suite also has a method named .toggle(). Which one is fired depends on the set of arguments passed.

With no parameters, the `.toggle()` method simply toggles the visibility of elements:

```
$('.target').toggle();
```

The matched elements will be revealed or hidden immediately, with no animation, by changing the CSS `display` property. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration is provided, `.toggle()` becomes an animation method. The `.toggle()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

We will cause `.toggle()` to be called when another element is clicked:

```
$('#clickme').click(function() {
 $('#book').toggle('slow', function() {
  // Animation complete.
 });
```

```
});
```

With the element initially shown, we can hide it slowly with the first click:

A second click will show the element once again:

The second version of the method accepts a Boolean parameter. If this parameter is `true`, then the matched elements are shown; if `false`, the elements are hidden. In essence, the statement:

```
$('#foo').toggle(showOrHide);
```

is equivalent to:

```
if ( showOrHide == true ) {
 $('#foo').show();
} else if ( showOrHide == false ) {
 $('#foo').hide();
}
```

Example
Toggles all paragraphs.

```
$("button").click(function () {
$("p").toggle();
});
```

Example
Animates all paragraphs to be shown if they are hidden and hidden if they are visible, completing the animation within 600 milliseconds.

```
$("button").click(function () {
$("p").toggle("slow");
});
```

Example
Shows all paragraphs, then hides them all, back and forth.

```
var flip = 0;
$("button").click(function () {
$("p").toggle( flip++ % 2 == 0 );
});
```

jQuery.isArray(obj)
Determine whether the argument is an array.

Arguments
obj - Object to test whether or not it is an array.

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

Example
Finds out if the parameter is an array.

```
$("b").append( "" + $.isArray([]) );
```

jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Rather than using `$.browser` to detect the current user agent and alter the page presentation based on which browser is running, it is a good practice to perform **feature detection**. This means that prior to executing code which relies on a browser feature, we test to ensure that the feature works properly. To make this process simpler, jQuery performs many such tests and makes the results available to us as properties of the `jQuery.support` object.

The values of all the support properties are determined using feature detection (and do not use any form of browser sniffing).

Following are a few resources that explain how feature detection works:

- http://peter.michaux.ca/articles/feature-detection-state-of-the-art-browser-scripting
- http://www.jibbering.com/faq/faq_notes/not_browser_detect.html
- http://yura.thinkweb2.com/cft/

While jQuery includes a number of properties, developers should feel free to add their own as their needs dictate. Many of the `jQuery.support` properties are rather low-level, so they are most useful for plugin and jQuery core development, rather than general day-to-day development. Since jQuery requires these tests internally, they must be performed on *every* page load; for that reason this list is kept short and limited to features needed by jQuery itself.

The tests included in `jQuery.support` are as follows:

- `ajax` is equal to true if a browser is able to create an `XMLHttpRequest` object.
- `boxModel` is equal to true if the page is rendering according to the W3C CSS Box Model (is currently false in IE 6 and 7 when they are in Quirks Mode). This property is null until document ready occurs.
- `changeBubbles` is equal to true if the change event bubbles up the DOM tree, as required by the W3C DOM event model. (It is currently false in IE, and jQuery simulates bubbling).
- `checkClone` is equal to true if a browser correctly clones the checked state of radio buttons or checkboxes in document fragments.
- `checkOn` is equal to true if the value of a checkbox defaults to "on" when no value is specified.
- `cors` is equal to true if a browser can create an `XMLHttpRequest` object and if that `XMLHttpRequest` object has a `withCredentials` property. To enable cross-domain requests in environments that do not support cors yet but do allow cross-domain XHR requests (windows gadget, etc), set `$.support.cors = true;`.CORS WD
- `cssFloat` is equal to true if the name of the property containing the CSS float value is .cssFloat, as defined in the CSS Spec. (It is currently false in IE, it uses styleFloat instead).
- `hrefNormalized` is equal to true if the `.getAttribute()` method retrieves the `href` attribute of elements unchanged, rather than normalizing it to a fully-qualified URL. (It is currently false in IE, the URLs are normalized). DOM l3 spec
- `htmlSerialize` is equal to true if the browser is able to serialize/insert `<link>` elements using the `.innerHTML` property of elements. (is currently false in IE). HTML5 WD
- `leadingWhitespace` is equal to true if the browser inserts content with .innerHTML exactly as provided-specifically, if leading whitespace characters are preserved. (It is currently false in IE 6-8). HTML5 WD
- `noCloneChecked` is equal to true if cloned DOM elements copy over the state of the `.checked` expando. (It is currently false in IE). (Added in jQuery 1.5.1)
- `noCloneEvent` is equal to true if cloned DOM elements are created without event handlers (that is, if the event handlers on the source element are not cloned). (It is currently false in IE). DOM l2 spec
- `opacity` is equal to true if a browser can properly interpret the opacity style property. (It is currently false in IE, it uses alpha filters instead). CSS3 spec
- `optDisabled` is equal to true if option elements within disabled select elements are not automatically marked as disabled. HTML5 WD
- `optSelected` is equal to true if an `<option>` element that is selected by default has a working `selected` property. HTML5 WD
- `scriptEval()` is equal to true if inline scripts are automatically evaluated and executed when inserted into the document using standard DOM manipulation methods such as `.appendChild()` and `.createTextNode()`. (It is currently false in IE, it uses `.text` to insert executable scripts). **Note: No longer supported; removed in jQuery 1.6. Prior to jQuery 1.5.1**, the `scriptEval()` method was the static `scriptEval` property. The change to a method allowed the test to be deferred until first use to prevent content security policy inline-script violations. HTML5 WD
- `style` is equal to true if inline styles for an element can be accessed through the DOM attribute called style, as required by the DOM Level 2 specification. In this case, `.getAttribute('style')` can retrieve this value; in Internet Explorer, `.cssText` is used for this purpose. DOM l2 Style spec
- `submitBubbles` is equal to true if the submit event bubbles up the DOM tree, as required by the W3C DOM event model. (It is currently false in IE, and jQuery simulates bubbling).
- `tbody` is equal to true if an empty `<table>` element can exist without a `<tbody>` element. According to the HTML specification, this sub-element is optional, so the property should be true in a fully-compliant browser. If false, we must account for the possibility of the browser injecting `<tbody>` tags implicitly. (It is currently false in IE, which automatically inserts `tbody` if it is not present in a string assigned to `innerHTML`). HTML5 spec

Example

Returns the box model for the iframe.

```
$("p").html("This frame uses the W3C box model: <span>" +
        jQuery.support.boxModel + "</span>");
```

Example
Returns false if the page is in QuirksMode in Internet Explorer

```
jQuery.support.boxModel
```

toggleClass(className)
Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Arguments
className - One or more class names (separated by spaces) to be toggled for each element in the matched set.

This method takes one or more class names as its parameter. In the first version, if an element in the matched set of elements already has the class, then it is removed; if an element does not have the class, then it is added. For example, we can apply `.toggleClass()` to a simple `<div>`:

```
<div class="tumble">Some text.</div>
```

The first time we apply `$('div.tumble').toggleClass('bounce')`, we get the following:

```
<div class="tumble bounce">Some text.</div>
```

The second time we apply `$('div.tumble').toggleClass('bounce')`, the `<div>` class is returned to the single `tumble` value:

```
<div class="tumble">Some text.</div>
```

Applying `.toggleClass('bounce spin')` to the same `<div>` alternates between `<div class="tumble bounce spin">` and `<div class="tumble">`.

The second version of `.toggleClass()` uses the second parameter for determining whether the class should be added or removed. If this parameter's value is `true`, then the class is added; if `false`, the class is removed. In essence, the statement:

```
$('#foo').toggleClass(className, addOrRemove);
```

is equivalent to:

```
if (addOrRemove) {
  $('#foo').addClass(className);
 }
 else {
  $('#foo').removeClass(className);
 }
```

As of jQuery 1.4, if no arguments are passed to `.toggleClass()`, all class names on the element the first time `.toggleClass()` is called will be toggled. Also as of jQuery 1.4, the class name to be toggled can be determined by passing in a function.

```
$('div.foo').toggleClass(function() {
 if ($(this).parent().is('.bar')) {
  return 'happy';
 } else {
  return 'sad';
 }
});
```

This example will toggle the `happy` class for `<div class="foo">` elements if their parent element has a class of `bar`; otherwise, it will toggle the `sad` class.

Example
Toggle the class 'highlight' when a paragraph is clicked.

```
$("p").click(function () {
   $(this).toggleClass("highlight");
  });
```

Example
Add the "highlight" class to the clicked paragraph on every third click of that paragraph, remove it every first and second click.

```
var count = 0;
$("p").each(function() {
 var $thisParagraph = $(this);
 var count = 0;
 $thisParagraph.click(function() {
  count++;
  $thisParagraph.find("span").text('clicks: ' + count);
  $thisParagraph.toggleClass("highlight", count % 3 == 0);
 });
});
```

Example
Toggle the class name(s) indicated on the buttons for each div.

```
var cls = ['', 'a', 'a b', 'a b c'];
var divs = $('div.wrap').children();
var appendClass = function() {
 divs.append(function() {
  return '<div>' + (this.className || 'none') + '</div>';
 });
};

appendClass();

$('button').bind('click', function() {
 var tc = this.className || undefined;
 divs.toggleClass(tc);
 appendClass();
});

$('a').bind('click', function(event) {
 event.preventDefault();
 divs.empty().each(function(i) {
  this.className = cls[i];
 });
 appendClass();
});
```

Version 1.4

jQuery.proxy(function, context)
Takes a function and returns a new one that will always have a particular context.

Arguments
function - The function whose context will be changed.
context - The object to which the context (`this`) of the function should be set.

This method is most useful for attaching event handlers to an element where the context is pointing back to a different object. Additionally, jQuery makes sure that even if you bind the function returned from `jQuery.proxy()` it will still unbind the correct function if passed the original.

Be aware, however, that jQuery's event binding subsystem assigns a unique id to each event handling function in order to track it when it is used to

specify the function to be unbound. The function represented by `jQuery.proxy()` is seen as a single function by the event subsystem, even when it is used to bind different contexts. To avoid unbinding the wrong handler, use a unique event namespace for binding and unbinding (e.g., `"click.myproxy1"`) rather than specifying the proxied function during unbinding.

Example
Change the context of functions bound to a click handler using the "function, context" signature. Unbind the first handler after first click.

```
var me = {
 type: "zombie",
 test: function(event) {
  // Without proxy, `this` would refer to the event target
  // use event.target to reference that element.
  var element = event.target;
  $(element).css("background-color", "red");

  // With proxy, `this` refers to the me object encapsulating
  // this function.
  $("#log").append( "Hello " + this.type + "<br>" );
  $("#test").unbind("click", this.test);
 }
};

var you = {
 type: "person",
 test: function(event) {
  $("#log").append( this.type + " " );
 }
};

// execute you.test() in the context of the `you` object
// no matter where it is called
// i.e. the `this` keyword will refer to `you`
var youClick = $.proxy( you.test, you );


// attach click handlers to #test
$("#test")
 // this === "zombie"; handler unbound after first click
 .click( $.proxy( me.test, me ) )
 // this === "person"
 .click( youClick )
 // this === "zombie"
 .click( $.proxy( you.test, me ) )
 // this === "<button> element"
 .click( you.test );
```

Example
Enforce the context of the function using the "context, function name" signature. Unbind the handler after first click.

```
var obj = {
  name: "John",
  test: function() {
   $("#log").append( this.name );
   $("#test").unbind("click", obj.test);
  }
 };

 $("#test").click( jQuery.proxy( obj, "test" ) );
```

focusout(handler(eventObject))
Bind an event handler to the "focusout" JavaScript event.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('focusout', handler)`.

The `focusout` event is sent to an element when it, or any element inside of it, loses focus. This is distinct from the blur event in that it supports detecting the loss of focus from parent elements (in other words, it supports event bubbling).

This event will likely be used together with the focusin event.

Example
Watch for a loss of focus to occur inside paragraphs and note the difference between the `focusout` count and the `blur` count.

```
var fo = 0, b = 0;
$("p").focusout(function() {
 fo++;
 $("#fo")
  .text("focusout fired: " + fo + "x");
}).blur(function() {
 b++;
 $("#b")
  .text("blur fired: " + b + "x");

});
```

focusin(handler(eventObject))
Bind an event handler to the "focusin" event.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('focusin', handler)`.

The `focusin` event is sent to an element when it, or any element inside of it, gains focus. This is distinct from the focus event in that it supports detecting the focus event on parent elements (in other words, it supports event bubbling).

This event will likely be used together with the focusout event.

Example
Watch for a focus to occur within the paragraphs on the page.

```
$("p").focusin(function() {
     $(this).find("span").css('display','inline').fadeOut(1000);
  });
```

has(selector)
Reduce the set of matched elements to those that have a descendant that matches the selector or DOM element.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.has()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against the descendants of the matching elements; the element will be included in the result if any of its descendant elements matches the selector.

Consider a page with a nested list as follows:

```
 <ul>
 <li>list item 1</li>
 <li>list item 2
```

```
  <ul>
   <li>list item 2-a</li>
   <li>list item 2-b</li>
  </ul>
 </li>
 <li>list item 3</li>
 <li>list item 4</li>
</ul>
```

We can apply this method to the set of list items as follows:

```
$('li').has('ul').css('background-color', 'red');
```

The result of this call is a red background for item 2, as it is the only `<li>` that has a `<ul>` among its descendants.

Example
Check if an element is inside another.

```
$("ul").append("<li>" + ($("ul").has("li").length ? "Yes" : "No") + "</li>");
 $("ul").has("li").addClass("full");
```

jQuery.contains(container, contained)
Check to see if a DOM element is within another DOM element.

Arguments
container - The DOM element that may contain the other element.
contained - The DOM element that may be contained by the other element.

Note: The first argument *must* be a DOM element, not a jQuery object or plain JavaScript object.

Example
Check if an element is inside another. Text and comment nodes are not supported.

```
jQuery.contains(document.documentElement, document.body); // true
jQuery.contains(document.body, document.documentElement); // false
```

jQuery.noop()
An empty function.

You can use this empty function when you wish to pass around a function that will do nothing.

This is useful for plugin authors who offer optional callbacks; in the case that no callback is given, something like `jQuery.noop` could execute.

delay(duration, [queueName])
Set a timer to delay execution of subsequent items in the queue.

Arguments
duration - An integer indicating the number of milliseconds to delay execution of the next item in the queue.
queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

Added to jQuery in version 1.4, the `.delay()` method allows us to delay the execution of functions that follow it in the queue. It can be used with the standard effects queue or with a custom queue. Only subsequent events in a queue are delayed; for example this will *not* delay the no-arguments forms of `.show()` or `.hide()` which do not use the effects queue.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of 200 and 600 milliseconds, respectively.

Using the standard effects queue, we can, for example, set an 800-millisecond delay between the `.slideUp()` and `.fadeIn()` of `<div`

id="foo">:

```
$('#foo').slideUp(300).delay(800).fadeIn(400);
```

When this statement is executed, the element slides up for 300 milliseconds and then pauses for 800 milliseconds before fading in for 400 milliseconds.

The `.delay()` method is best for delaying between queued jQuery effects. Because it is limited-it doesn't, for example, offer a way to cancel the delay-`.delay()` is not a replacement for JavaScript's native setTimeout function, which may be more appropriate for certain use cases.

Example
Animate the hiding and showing of two divs, delaying the first before showing it.

```
$("button").click(function() {
   $("div.first").slideUp(300).delay(800).fadeIn(400);
   $("div.second").slideUp(300).fadeIn(400);
  });
```

parentsUntil([selector], [filter])
Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

Arguments
selector - A string containing a selector expression to indicate where to stop matching ancestor elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.parentsUntil()` method traverses through the ancestors of these elements until it reaches an element matched by the selector passed in the method's argument. The resulting jQuery object contains all of the ancestors up to but not including the one matched by the `.parentsUntil()` selector.

If the selector is not matched or is not supplied, all ancestors will be selected; in these cases it selects the same elements as the `.parents()` method does when no selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be used for the first **.parentsUntil()** argument.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

Example
Find the ancestors of <li class="item-a"> up to <ul class="level-1"> and give them a red background color. Also, find ancestors of <li class="item-2"> that have a class of "yes" up to <ul class="level-1"> and give them a green border.

```
$("li.item-a").parentsUntil(".level-1")
 .css("background-color", "red");

$("li.item-2").parentsUntil( $("ul.level-1"), ".yes" )
 .css("border", "3px solid green");
```

prevUntil([selector], [filter])
Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

Arguments
selector - A string containing a selector expression to indicate where to stop matching preceding sibling elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.prevUntil()` method searches through the predecessors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all previous siblings up to but not including the one matched by the `.prevUntil()` selector; the elements are returned in order from the closest sibling to the farthest.

If the selector is not matched or is not supplied, all previous siblings will be selected; in these cases it selects the same elements as the `.prevAll()`

method does when no filter selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be used for the first **.prevUntil()** argument.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

Example
Find the siblings that precede <dt id="term-2"> up to the preceding <dt> and give them a red background color. Also, find previous <dd> siblings of <dt id="term-3"> up to <dt id="term-1"> and give them a green text color.

```
$("#term-2").prevUntil("dt")
 .css("background-color", "red");

var term1 = document.getElementById('term-1');
$("#term-3").prevUntil(term1, "dd")
 .css("color", "green");
```

nextUntil([selector], [filter])
Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

Arguments
selector - A string containing a selector expression to indicate where to stop matching following sibling elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.nextUntil()` method searches through the successors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all following siblings up to but not including the one matched by the `.nextUntil()` argument.

If the selector is not matched or is not supplied, all following siblings will be selected; in these cases it selects the same elements as the `.nextAll()` method does when no filter selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be passed to the `.nextUntil()` method.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

Example
Find the siblings that follow <dt id="term-2"> up to the next <dt> and give them a red background color. Also, find <dd> siblings that follow <dt id="term-1"> up to <dt id="term-3"> and give them a green text color.

```
$("#term-2").nextUntil("dt")
 .css("background-color", "red");

var term3 = document.getElementById("term-3");
$("#term-1").nextUntil(term3, "dd")
 .css("color", "green");
```

jQuery.data(element, key, value)
Store arbitrary data associated with the specified element. Returns the value that was set.

Arguments
element - The DOM element to associate with the data.
key - A string naming the piece of data to set.
value - The new data value.

Note: This is a low-level method; a more convenient `.data()` is also available.

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');
```

Note: 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.

Example
Store then retrieve a value from the div element.

```
var div = $("div")[0];
  jQuery.data(div, "test", { first: 16, last: "pizza!" });
  $("span:first").text(jQuery.data(div, "test").first);
  $("span:last").text(jQuery.data(div, "test").last);
```

jQuery.data(element, key)
Returns value at named data store for the element, as set by `jQuery.data(element, name, value)`, or the full data store for the element.

Arguments
element - The DOM element to query for the data.
key - Name of the data stored.

Note: This is a low-level method; a more convenient `.data()` is also available.

Regarding HTML5 data-* attributes: This low-level method does NOT retrieve the `data-*` attributes unless the more convenient `.data()` method has already retrieved them.

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 from memory leaks. We can retrieve several distinct values for a single element one at a time, or as a set:

```
alert(jQuery.data( document.body, 'foo' ));
alert(jQuery.data( document.body ));
```

The above lines alert the data values that were set on the `body` element. If nothing was set on that element, an empty string is returned.

Calling `jQuery.data(element)` retrieves all of the element's associated values as a JavaScript object. Note that jQuery itself uses this method to store data for internal use, such as event handlers, so do not assume that it contains only data that your own code has stored.

Note: 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.

Example
Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value, div = $("div")[0];

 switch ($("button").index(this)) {
  case 0 :
   value = jQuery.data(div, "blah");
   break;
  case 1 :
   jQuery.data(div, "blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   jQuery.data(div, "blah", 86);
   value = "Stored!";
   break;
  case 3 :
   jQuery.removeData(div, "blah");
```

```
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

clearQueue([queueName])

Remove from the queue all items that have not yet been run.

Arguments

queueName - A string containing the name of the queue. Defaults to `fx`, the standard effects queue.

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.

Example

Empty the queue.

```
$("#start").click(function () {

 var myDiv = $("div");
 myDiv.show("slow");
 myDiv.animate({left:'+=200'},5000);
 myDiv.queue(function () {
  var _this = $(this);
  _this.addClass("newcolor");
  _this.dequeue();
 });

 myDiv.animate({left:'-=200'},1500);
 myDiv.queue(function () {
  var _this = $(this);
  _this.removeClass("newcolor");
  _this.dequeue();
 });
 myDiv.slideUp();

});

$("#stop").click(function () {
 var myDiv = $("div");
 myDiv.clearQueue();
 myDiv.stop();
});
```

toArray()

Retrieve all the DOM elements contained in the jQuery set, as an array.

`.toArray()` returns all of the elements in the jQuery set:

```
alert($('li').toArray());
```

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

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

Example

Selects all divs in the document and returns the DOM Elements as an Array, then uses the built-in reverse-method to reverse that array.

```
function disp(divs) {
   var a = [];
   for (var i = 0; i < divs.length; i++) {
    a.push(divs[i].innerHTML);
   }
   $("span").text(a.join(" "));
  }

  disp( $("div").toArray().reverse() );
```

jQuery.isEmptyObject(object)

Check to see if an object is empty (contains no properties).

Arguments

object - The object that will be checked to see if it's empty.

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()

Example

Check an object to see if it's empty.

```
jQuery.isEmptyObject({}) // true
jQuery.isEmptyObject({ foo: "bar" }) // false
```

jQuery.isPlainObject(object)

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

Arguments

object - The object that will be checked to see if it's a plain 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.

An example of this is a test against `document.location` using `$.isPlainObject()` as follows:

```
console.log($.isPlainObject(document.location));
```

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. Some basic example of use-cases that do function correctly cross-browser can be found below.

Example

Check an object to see if it's a plain object.

```
jQuery.isPlainObject({}) // true
jQuery.isPlainObject("test") // false
```

index()

Search for a given element from among the matched elements.

Return Values

If no argument is passed to the `.index()` method, the return value is an integer indicating the position of the first element within the jQuery object relative to its sibling elements.

If `.index()` is called on a collection of elements and a DOM element or jQuery object is passed in, `.index()` returns an integer indicating the position of the passed element relative to the original collection.

If a selector string is passed as an argument, `.index()` returns an integer indicating the position of the original element relative to the elements matched by the selector. If the element is not found, `.index()` will return -1.

Detail

The complementary operation to `.get()`, which accepts an index and returns a DOM node, `.index()` can take a DOM node and returns an index. Suppose we have 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 retrieve 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');
alert('Index: ' + $('li').index(listItem));
We get back the zero-based position of the list item:
```

Index: 1

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

```
var listItem = $('#bar');
alert('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)');
alert('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 which also matches this selector is located.

```
var listItem = $('#bar');
alert('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:

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

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

Index: 1

Example
On click, returns the index (based zero) of that div in the page.

```
$("div").click(function () {
 // this is the dom element clicked
 var index = $("div").index(this);
 $("span").text("That was div index #" + index);
});
```

Example
Returns the index for the element with ID bar.

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

Example
Returns the index for the first item in the jQuery collection.

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

Example
Returns the index for the element with ID bar in relation to all <li> elements.

```
$('div').html('Index: ' + $('#bar').index('li') );
```

Example
Returns the index for the element with ID bar in relation to its siblings.

```
var barIndex = $('#bar').index();
$('div').html( 'Index: ' + barIndex );
```

Example
Returns -1, as there is no element with ID foobar.

```
var foobar = $("li").index( $('#foobar') );
$('div').html('Index: ' + foobar);
```

data(key, value)
Store arbitrary data associated with the matched elements.

Arguments
key - A string naming the piece of data to set.
value - The new data value; it can be any Javascript type including Array or Object.

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 later:

```
$('body').data('foo', 52);
$('body').data('bar', { myType: 'test', count: 40 });

$('body').data('foo'); // 52
$('body').data(); // {foo: 52, bar: { myType: 'test', count: 40 }}
```

In jQuery 1.4.3 setting an element's data object with `.data(obj)` extends the data previously stored with that element. jQuery itself uses the `.data()` method to save information under the names 'events' and 'handle', and also reserves any data name starting with an underscore ('_') for internal use.

Prior to jQuery 1.4.3 (starting in jQuery 1.4) the .data() method completely replaced all data, instead of just extending the data object. If you are using third-party plugins it may not be advisable to completely replace the element's data object, since plugins may have also set data.

Due to the way browsers interact with plugins and external code, the `.data()` method cannot be used on `<object>` (unless it's a Flash plugin), `<applet>` or `<embed>` elements.

Example
Store then retrieve a value from the div element.

```
$("div").data("test", { first: 16, last: "pizza!" });
$("span:first").text($("div").data("test").first);
$("span:last").text($("div").data("test").last);
```

data(key)
Returns value at named data store for the first element in the jQuery collection, as set by data(name, value).

Arguments
key - Name of the data stored.

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 retrieve several distinct values for a single element one at a time, or as a set:

```
alert($('body').data('foo'));
alert($('body').data());
```

The above lines alert the data values that were set on the `body` element. If no data at all was set on that element, `undefined` is returned.

```
alert( $("body").data("foo")); //undefined
$("body").data("bar", "foobar");
alert( $("body").data("bar")); //foobar
```

HTML5 data-* Attributes

As of jQuery 1.4.3 HTML 5 data- attributes will be automatically pulled in to jQuery's data object. The treatment of attributes with embedded dashes was changed in jQuery 1.6 to conform to the W3C HTML5 specification.

For example, given the following HTML:

```
<div data-role="page" data-last-value="43" data-hidden="true" data-options='{"name":"John"}'></div>
```

All of the following jQuery code will work.

```
$("div").data("role") === "page";
$("div").data("lastValue") === 43;
$("div").data("hidden") === true;
```

```
$("div").data("options").name === "John";
```

Every attempt is made to convert the string to a JavaScript value (this includes booleans, numbers, objects, arrays, and null) otherwise it is left as a string. To retrieve the value's attribute as a string without any attempt to convert it, use the `attr()` method. When the data attribute is an object (starts with '{') or array (starts with '[') then `jQuery.parseJSON` is used to parse the string; it must follow valid JSON syntax*including quoted property names*. The data- attributes are pulled in the first time the data property is accessed and then are no longer accessed or mutated (all data values are then stored internally in jQuery).

Calling `.data()` with no parameters retrieves all of the values as a JavaScript object. This object can be safely cached in a variable as long as a new object is not set with `.data(obj)`. Using the object directly to get or set values is faster than making individual calls to `.data()` to get or set each value:

```
var mydata = $("#mydiv").data();
if ( mydata.count < 9 ) {
  mydata.count = 43;
  mydata.status = "embiggened";
}
```

Example

Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value;

 switch ($("button").index(this)) {
  case 0 :
   value = $("div").data("blah");
   break;
  case 1 :
   $("div").data("blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   $("div").data("blah", 86);
   value = "Stored!";
   break;
  case 3 :
   $("div").removeData("blah");
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

bind(eventType, [eventData], handler(eventObject))

Attach a handler to an event for the elements.

Arguments

eventType - A string containing one or more DOM event types, such as "click" or "submit," or custom event names.
eventData - A map of data that will be passed to the event handler.
handler(eventObject) - A function to execute each time the event is triggered.

As of jQuery 1.7, the .on() method is the preferred method for attaching event handlers to a document. For earlier versions, the `.bind()` method is used for attaching an event handler directly to elements. Handlers are attached to the currently selected elements in the jQuery object, so those elements *must exist* at the point the call to `.bind()` occurs. For more flexible event binding, see the discussion of event delegation in .on() or .delegate().

Any string is legal for `eventType`; if the string is not the name of a native DOM event, then the handler is bound to a custom event. These events are never called by the browser, but may be triggered manually from other JavaScript code using `.trigger()` or `.triggerHandler()`.

If the `eventType` string contains a period (`.`) character, then the event is namespaced. The period character separates the event from its namespace. For example, in the call `.bind('click.name', handler)`, the string `click` is the event type, and the string `name` is the namespace. Namespacing allows us to unbind or trigger some events of a type without affecting others. See the discussion of `.unbind()` for more information.

There are shorthand methods for some standard browser events such as .click() that can be used to attach or trigger event handlers. For a complete list of shorthand methods, see the events category.

When an event reaches an element, all handlers bound to that event type for the element are fired. If there are multiple handlers registered, they will always execute in the order in which they were bound. After all handlers have executed, the event continues along the normal event propagation path.

A basic usage of `.bind()` is:

```
$('#foo').bind('click', function() {
 alert('User clicked on "foo."');
});
```

This code will cause the element with an ID of `foo` to respond to the `click` event. When a user clicks inside this element thereafter, the alert will be shown.

Multiple Events

Multiple event types can be bound at once by including each one separated by a space:

```
$('#foo').bind('mouseenter mouseleave', function() {
 $(this).toggleClass('entered');
});
```

The effect of this on `<div id="foo">` (when it does not initially have the "entered" class) is to add the "entered" class when the mouse enters the `<div>` and remove the class when the mouse leaves.

As of jQuery 1.4 we can bind multiple event handlers simultaneously by passing a map of event type/handler pairs:

```
$('#foo').bind({
 click: function() {
  // do something on click
 },
 mouseenter: function() {
  // do something on mouseenter
 }
});
```

Event Handlers

The `handler` parameter takes a callback function, as shown above. Within the handler, the keyword `this` refers to the DOM element to which the handler is bound. To make use of the element in jQuery, it can be passed to the normal `$()` function. For example:

```
$('#foo').bind('click', function() {
 alert($(this).text());
});
```

After this code is executed, when the user clicks inside the element with an ID of `foo`, its text contents will be shown as an alert.

As of jQuery 1.4.2 duplicate event handlers can be bound to an element instead of being discarded. This is useful when the event data feature is being used, or when other unique data resides in a closure around the event handler function.

In jQuery 1.4.3 you can now pass in `false` in place of an event handler. This will bind an event handler equivalent to: `function(){ return false; }`. This function can be removed at a later time by calling: `.unbind( eventName, false )`.

The Event object

The `handler` callback function can also take parameters. When the function is called, the event object will be passed to the first parameter.

The event object is often unnecessary and the parameter omitted, as sufficient context is usually available when the handler is bound to know exactly what needs to be done when the handler is triggered. However, at times it becomes necessary to gather more information about the user's environment at the time the event was initiated. View the full Event Object.

Returning `false` from a handler is equivalent to calling both `.preventDefault()` and `.stopPropagation()` on the event object.

Using the event object in a handler looks like this:

```
$(document).ready(function() {
 $('#foo').bind('click', function(event) {
  alert('The mouse cursor is at ('
   + event.pageX + ', ' + event.pageY + ')');
 });
});
```

Note the parameter added to the anonymous function. This code will cause a click on the element with ID `foo` to report the page coordinates of the mouse cursor at the time of the click.

Passing Event Data

The optional `eventData` parameter is not commonly used. When provided, this argument allows us to pass additional information to the handler. One handy use of this parameter is to work around issues caused by closures. For example, suppose we have two event handlers that both refer to the same external variable:

```
var message = 'Spoon!';
$('#foo').bind('click', function() {
 alert(message);
});
message = 'Not in the face!';
$('#bar').bind('click', function() {
 alert(message);
});
```

Because the handlers are closures that both have `message` in their environment, both will display the message Not in the face! when triggered. The variable's value has changed. To sidestep this, we can pass the message in using `eventData`:

```
var message = 'Spoon!';
$('#foo').bind('click', {msg: message}, function(event) {
 alert(event.data.msg);
});
message = 'Not in the face!';
$('#bar').bind('click', {msg: message}, function(event) {
 alert(event.data.msg);
});
```

This time the variable is not referred to directly within the handlers; instead, the variable is passed in *by value* through `eventData`, which fixes the value at the time the event is bound. The first handler will now display Spoon! while the second will alert Not in the face!

Note that objects are passed to functions *by reference*, which further complicates this scenario.

If `eventData` is present, it is the second argument to the `.bind()` method; if no additional data needs to be sent to the handler, then the callback is

passed as the second and final argument.

See the `.trigger()` method reference for a way to pass data to a handler at the time the event happens rather than when the handler is bound.

As of jQuery 1.4 we can no longer attach data (and thus, events) to object, embed, or applet elements because critical errors occur when attaching data to Java applets.

Note: Although demonstrated in the next example, it is inadvisable to bind handlers to both the `click` and `dblclick` events for the same element. The sequence of events triggered varies from browser to browser, with some receiving two click events before the `dblclick` and others only one. Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

Example
Handle click and double-click for the paragraph. Note: the coordinates are window relative, so in this case relative to the demo iframe.

```
$("p").bind("click", function(event){
var str = "( " + event.pageX + ", " + event.pageY + " )";
$("span").text("Click happened! " + str);
});
$("p").bind("dblclick", function(){
$("span").text("Double-click happened in " + this.nodeName);
});
$("p").bind("mouseenter mouseleave", function(event){
$(this).toggleClass("over");
});
```

Example
To display each paragraph's text in an alert box whenever it is clicked:

```
$("p").bind("click", function(){
alert( $(this).text() );
});
```

Example
You can pass some extra data before the event handler:

```
function handler(event) {
alert(event.data.foo);
}
$("p").bind("click", {foo: "bar"}, handler)
```

Example
Cancel a default action and prevent it from bubbling up by returning `false`:

```
$("form").bind("submit", function() { return false; })
```

Example
Cancel only the default action by using the .preventDefault() method.

```
$("form").bind("submit", function(event) {
event.preventDefault();
});
```

Example
Stop an event from bubbling without preventing the default action by using the .stopPropagation() method.

```
$("form").bind("submit", function(event) {
 event.stopPropagation();
});
```

Example
Bind custom events.

```
$("p").bind("myCustomEvent", function(e, myName, myValue){
$(this).text(myName + ", hi there!");
$("span").stop().css("opacity", 1)
.text("myName = " + myName)
.fadeIn(30).fadeOut(1000);
});
$("button").click(function () {
$("p").trigger("myCustomEvent", [ "John" ]);
});
```

Example

Bind multiple events simultaneously.

```
$("div.test").bind({
 click: function(){
  $(this).addClass("active");
 },
 mouseenter: function(){
  $(this).addClass("inside");
 },
 mouseleave: function(){
  $(this).removeClass("inside");
 }
});
```

first()

Reduce the set of matched elements to the first in the set.

[

Given a jQuery object that represents a set of DOM elements, the `.first()` method constructs a new jQuery object from the first matching element.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').first().css('background-color', 'red');
```

The result of this call is a red background for the first item.

Example

Highlight the first span in a paragraph.

```
$("p span").first().addClass('highlight');
```

last()

Reduce the set of matched elements to the final one in the set.

[

Given a jQuery object that represents a set of DOM elements, the `.last()` method constructs a new jQuery object from the last matching element.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').last().css('background-color', 'red');
```

The result of this call is a red background for the final item.

Example

Highlight the last span in a paragraph.

```
$("p span").last().addClass('highlight');
```

jQuery(selector, [context])

Accepts a string containing a CSS selector which is then used to match a set of elements.

Arguments

selector - A string containing a selector expression
context - A DOM Element, Document, or jQuery to use as context

In the first formulation listed above, `jQuery()` - which can also be written as `$()` - searches through the DOM for any elements that match the provided selector and creates a new jQuery object that references these elements:

```
$('div.foo');
```

If no elements match the provided selector, the new jQuery object is "empty"; that is, it contains no elements and has `.length` property of 0.

Selector Context

By default, selectors perform their searches within the DOM starting at the document root. However, an alternate context can be given for the search by using the optional second parameter to the `$()` function. For example, to do a search within an event handler, the search can be restricted like so:

```
$('div.foo').click(function() {
 $('span', this).addClass('bar');
});
```

When the search for the span selector is restricted to the context of `this`, only spans within the clicked element will get the additional class.

Internally, selector context is implemented with the `.find()` method, so `$('span', this)` is equivalent to `$(this).find('span')`.

Using DOM elements

The second and third formulations of this function create a jQuery object using one or more DOM elements that were already selected in some other way. A common use of this facility is to call jQuery methods on an element that has been passed to a callback function through the keyword `this`:

```
$('div.foo').click(function() {
 $(this).slideUp();
});
```

This example causes elements to be hidden with a sliding animation when clicked. Because the handler receives the clicked item in the `this` keyword as a bare DOM element, the element must be passed to the `$()` function before applying jQuery methods to it.

XML data returned from an Ajax call can be passed to the `$()` function so individual elements of the XML structure can be retrieved using `.find()` and other DOM traversal methods.

```
$.post('url.xml', function(data) {
 var $child = $(data).find('child');
})
```

Cloning jQuery Objects

When a jQuery object is passed to the `$()` function, a clone of the object is created. This new jQuery object references the same DOM elements as the initial one.

Returning an Empty Set

As of jQuery 1.4, calling the `jQuery()` method with *no arguments* returns an empty jQuery set (with a `.length` property of 0). In previous versions of jQuery, this would return a set containing the document node.

Working With Plain Objects

At present, the only operations supported on plain JavaScript objects wrapped in jQuery are: `.data()`,`.prop()`,`.bind()`, `.unbind()`, `.trigger()` and `.triggerHandler()`. The use of `.data()` (or any method requiring `.data()`) on a plain object will result in a new property on the object called jQuery{randomNumber} (eg. jQuery123456789).

```
// define a plain object
var foo = {foo:'bar', hello:'world'};

// wrap this with jQuery
var $foo = $(foo);

// test accessing property values
var test1 = $foo.prop('foo'); // bar

// test setting property values
$foo.prop('foo', 'foobar');
var test2 = $foo.prop('foo'); // foobar

// test using .data() as summarized above
$foo.data('keyName', 'someValue');
console.log($foo); // will now contain a jQuery{randomNumber} property

// test binding an event name and triggering
$foo.bind('eventName', function (){
    console.log('eventName was called');
});

$foo.trigger('eventName'); // logs 'eventName was called'
```

Should `.trigger('eventName')` be used, it will search for an 'eventName' property on the object and attempt to execute it after any attached jQuery handlers are executed. It does not check whether the property is a function or not. To avoid this behavior, `.triggerHandler('eventName')` should be used instead.

```
$foo.triggerHandler('eventName'); // also logs 'eventName was called'
```

Example
Find all p elements that are children of a div element and apply a border to them.

```
$("div > p").css("border", "1px solid gray");
```

Example
Find all inputs of type radio within the first form in the document.

```
$("input:radio", document.forms[0]);
```

Example
Find all div elements within an XML document from an Ajax response.

```
$("div", xml.responseXML);
```

Example
Set the background color of the page to black.

```
$(document.body).css( "background", "black" );
```

Example
Hide all the input elements within a form.

```
$(myForm.elements).hide()
```

jQuery(html, [ownerDocument])
Creates DOM elements on the fly from the provided string of raw HTML.

Arguments
html - A string of HTML to create on the fly. Note that this parses HTML, **not** XML.
ownerDocument - A document in which the new elements will be created

Creating New Elements

If a string is passed as the parameter to `$()`, jQuery examines the string to see if it looks like HTML (i.e., it has `<tag ... >` somewhere within the string). If not, the string is interpreted as a selector expression, as explained above. But if the string appears to be an HTML snippet, jQuery attempts to create new DOM elements as described by the HTML. Then a jQuery object is created and returned that refers to these elements. You can perform any of the usual jQuery methods on this object:

```
$('<p id="test">My <em>new</em> text</p>').appendTo('body');
```

If the HTML is more complex than a single tag without attributes, as it is in the above example, the actual creation of the elements is handled by the browser's `innerHTML` mechanism. In most cases, jQuery creates a new element and sets the innerHTML property of the element to the HTML snippet that was passed in. When the parameter has a single tag, such as `$('<img />')` or `$('<a></a>')`, jQuery creates the element using the native JavaScript `createElement()` function.

When passing in complex HTML, some browsers may not generate a DOM that exactly replicates the HTML source provided. As mentioned, we use the browser's `.innerHTML` property to parse the passed HTML and insert it into the current document. During this process, some browsers filter out certain elements such as `<html>`, `<title>`, or <head> elements. As a result, the elements inserted may not be representative of the original string passed.

Filtering isn't however just limited to these tags. For example, Internet Explorer prior to version 8 will also convert all `href` properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate compatibility layer.

To ensure cross-platform compatibility, the snippet must be well-formed. Tags that can contain other elements should be paired with a closing tag:

```
$('<a href="http://jquery.com"></a>');
```

Alternatively, jQuery allows XML-like tag syntax (with or without a space before the slash):

```
$('<a/>');
```

Tags that cannot contain elements may be quick-closed or not:

```
$('<img />');
$('<input>');
```

When passing HTML to `jQuery()`, please also note that text nodes are not treated as DOM elements. With the exception of a few methods (such as `.content()`), they are generally otherwise ignored or removed. E.g:

```
var el = $('1<br/>2<br/>3'); // returns [<br>, "2", <br>]
el = $('1<br/>2<br/>3 >'); // returns [<br>, "2", <br>, "3 &gt;"]
```

This behaviour is expected.

As of jQuery 1.4, the second argument to `jQuery()` can accept a map consisting of a superset of the properties that can be passed to the .attr() method. Furthermore, any event type can be passed in, and the following jQuery methods can be called: val, css, html, text, data, width, height, or offset. The name `"class"` must be quoted in the map since it is a JavaScript reserved word, and `"className"` cannot be used since it is not the correct attribute name.

Note: Internet Explorer will not allow you to create an `input` or `button` element and change its type; you must specify the type using `'<input type="checkbox" />'` for example. A demonstration of this can be seen below:

Unsupported in IE:

```
$('<input />', {
  type: 'text',
  name: 'test'
}).appendTo("body");
```

Supported workaround:

```
$('<input type="text" />').attr({
  name: 'test'
}).appendTo("body");
```

Example

Create a div element (and all of its contents) dynamically and append it to the body element. Internally, an element is created and its innerHTML property set to the given markup.

```
$("<div><p>Hello</p></div>").appendTo("body")
```

Example

Create some DOM elements.

```
$("<div/>", {
 "class": "test",
 text: "Click me!",
 click: function(){
  $(this).toggleClass("test");
 }
}).appendTo("body");
```

jQuery(callback)

Binds a function to be executed when the DOM has finished loading.

Arguments

callback - The function to execute when the DOM is ready.

This function behaves just like `$(document).ready()`, in that it should be used to wrap other `$()` operations on your page that depend on the DOM being ready. While this function is, technically, chainable, there really isn't much use for chaining against it.

Example
Execute the function when the DOM is ready to be used.

```
$(function(){
  // Document is ready
 });
```

Example
Use both the shortcut for $(document).ready() and the argument to write failsafe jQuery code using the $ alias, without relying on the global alias.

```
jQuery(function($) {
  // Your code using failsafe $ alias here...
 });
```

closest(selector)
Get the first element that matches the selector, beginning at the current element and progressing up through the DOM tree.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.closest()` method searches through these elements and their ancestors in the DOM tree and constructs a new jQuery object from the matching elements. The `.parents()` and `.closest()` methods are similar in that they both traverse up the DOM tree. The differences between the two, though subtle, are significant:

.closest()	.parents()
Begins with the current element	Begins with the parent element
Travels up the DOM tree until it finds a match for the supplied selector	Travels up the DOM tree to the document's root element, adding each ancestor element to a temporary collectio
The returned jQuery object contains zero or one element	The returned jQuery object contains zero, one, or multiple elements

```
<ul id="one" class="level-1">
 <li class="item-i">I</li>
 <li id="ii" class="item-ii">II
 <ul class="level-2">
  <li class="item-a">A</li>
  <li class="item-b">B
   <ul class="level-3">
    <li class="item-1">1</li>
    <li class="item-2">2</li>
    <li class="item-3">3</li>
   </ul>
  </li>
  <li class="item-c">C</li>
 </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

Suppose we perform a search for `<ul>` elements starting at item A:

```
$('li.item-a').closest('ul')
 .css('background-color', 'red');
```

This will change the color of the level-2 <ul>, since it is the first encountered when traveling up the DOM tree.

Suppose we search for an <li> element instead:

```
$('li.item-a').closest('li')
 .css('background-color', 'red');
```

This will change the color of list item A. The .closest() method begins its search *with the element itself* before progressing up the DOM tree, and stops when item A matches the selector.

We can pass in a DOM element as the context within which to search for the closest element.

```
var listItemII = document.getElementById('ii');
$('li.item-a').closest('ul', listItemII)
 .css('background-color', 'red');
$('li.item-a').closest('#one', listItemII)
 .css('background-color', 'green');
```

This will change the color of the level-2 <ul>, because it is both the first <ul> ancestor of list item A and a descendant of list item II. It will not change the color of the level-1 <ul>, however, because it is not a descendant of list item II.

Example
Show how event delegation can be done with closest. The closest list element toggles a yellow background when it or its descendent is clicked.

```
$( document ).bind("click", function( e ) {
  $( e.target ).closest("li").toggleClass("hilight");
 });
```

Example
Pass a jQuery object to closest. The closest list element toggles a yellow background when it or its descendent is clicked.

```
var $listElements = $("li").css("color", "blue");
 $( document ).bind("click", function( e ) {
  $( e.target ).closest( $listElements ).toggleClass("hilight");
 });
```

closest(selectors, [context])
Gets an array of all the elements and selectors matched against the current element up through the DOM tree.

Arguments
selectors - An array or string containing a selector expression to match elements against (can also be a jQuery object).
context - A DOM element within which a matching element may be found. If no context is passed in then the context of the jQuery set will be used instead.

This signature (only!) is deprecated as of jQuery 1.7. This method is primarily meant to be used internally or by plugin authors.

Example
Show how event delegation can be done with closest.

```
var close = $("li:first").closest(["ul", "body"]);
 $.each(close, function(i){
 $("li").eq(i).html( this.selector + ": " + this.elem.nodeName );
 });
```

add(selector)
Add elements to the set of matched elements.

Arguments
selector - A string representing a selector expression to find additional elements to add to the set of matched elements.

Given a jQuery object that represents a set of DOM elements, the `.add()` method constructs a new jQuery object from the union of those elements and the ones passed into the method. The argument to `.add()` can be pretty much anything that `$()` accepts, including a jQuery selector expression, references to DOM elements, or an HTML snippet.

The updated set of elements can be used in a following (chained) method, or assigned to a variable for later use. For example:

```
$("p").add("div").addClass("widget");
var pdiv = $("p").add("div");
```

The following will *not* save the added elements, because the `.add()` method creates a new set and leaves the original set in pdiv unchanged:

```
var pdiv = $("p");
pdiv.add("div"); // WRONG, pdiv will not change
```

Consider a page with a simple list and a paragraph following it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
</ul>
<p>a paragraph</p>
```

We can select the list items and then the paragraph by using either a selector or a reference to the DOM element itself as the `.add()` method's argument:

```
$('li').add('p').css('background-color', 'red');
```

Or:

```
$('li').add(document.getElementsByTagName('p')[0])
 .css('background-color', 'red');
```

The result of this call is a red background behind all four elements.Using an HTML snippet as the `.add()` method's argument (as in the third version), we can create additional elements on the fly and add those elements to the matched set of elements. Let's say, for example, that we want to alter the background of the list items along with a newly created paragraph:

```
$('li').add('<p id="new">new paragraph</p>')
 .css('background-color', 'red');
```

Although the new paragraph has been created and its background color changed, it still does not appear on the page. To place it on the page, we could add one of the insertion methods to the chain.

As of jQuery 1.4 the results from .add() will always be returned in document order (rather than a simple concatenation).

Note: To reverse the `.add()` you can use .not(elements | selector) to remove elements from the jQuery results, or .end() to return to the selection before you added.

Example

Finds all divs and makes a border. Then adds all paragraphs to the jQuery object to set their backgrounds yellow.

```
$("div").css("border", "2px solid red")
    .add("p")
    .css("background", "yellow");
```

Example

Adds more elements, matched by the given expression, to the set of matched elements.

```
$("p").add("span").css("background", "yellow");
```

Example
Adds more elements, created on the fly, to the set of matched elements.

```
$("p").clone().add("<span>Again</span>").appendTo(document.body);
```

Example
Adds one or more Elements to the set of matched elements.

```
$("p").add(document.getElementById("a")).css("background", "yellow");
```

Example
Demonstrates how to add (or push) elements to an existing collection

```
var collection = $("p");
// capture the new collection
collection = collection.add(document.getElementById("a"));
collection.css("background", "yellow");
```

not(selector)
Remove elements from the set of matched elements.

Arguments
selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.not()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; the elements that don't match the selector will be included in the result.

Consider a page with a simple list on it:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can apply this method to the set of list items:

```
$('li').not(':even').css('background-color', 'red');
```

The result of this call is a red background for items 2 and 4, as they do not match the selector (recall that :even and :odd use 0-based indexing).

Removing Specific Elements

The second version of the `.not()` method allows us to remove elements from the matched set, assuming we have found those elements previously by some other means. For example, suppose our list had an id applied to one of its items:

```
<ul>
 <li>list item 1</li>
 <li>list item 2</li>
 <li id="notli">list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

We can fetch the third list item using the native JavaScript `getElementById()` function, then remove it from a jQuery object:

```
$('li').not(document.getElementById('notli'))
 .css('background-color', 'red');
```

This statement changes the color of items 1, 2, 4, and 5. We could have accomplished the same thing with a simpler jQuery expression, but this technique can be useful when, for example, other libraries provide references to plain DOM nodes.

As of jQuery 1.4, the `.not()` method can take a function as its argument in the same way that `.filter()` does. Elements for which the function returns `true` are excluded from the filtered set; all other elements are included.

Example
Adds a border to divs that are not green or blue.

```
$("div").not(".green, #blueone")
      .css("border-color", "red");
```

Example
Removes the element with the ID "selected" from the set of all paragraphs.

```
$("p").not( $("#selected")[0] )
```

Example
Removes the element with the ID "selected" from the set of all paragraphs.

```
$("p").not("#selected")
```

Example
Removes all elements that match "div p.selected" from the total set of all paragraphs.

```
$("p").not($("div p.selected"))
```

jQuery.param(obj)
Create a serialized representation of an array or object, suitable for use in a URL query string or Ajax request.

Arguments
obj - An array or object to serialize.

This function is used internally to convert form element values into a serialized string representation (See .serialize() for more information).

As of jQuery 1.3, the return value of a function is used instead of the function as a String.

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. You can disable this functionality globally by setting `jQuery.ajaxSettings.traditional = true;`.

If the object passed is in an Array, it must be an array of objects in the format returned by .serializeArray()

```
[{name:"first",value:"Rick"},
{name:"last",value:"Astley"},
{name:"job",value:"Rock Star"}]
```

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

Note: Because there is no universally agreed-upon specification for param strings, it is not possible to encode complex data structures using this method in a manner that works ideally across all languages supporting such input. Until such time that there is, the `$.param` method will remain in its current form.

In jQuery 1.4, HTML5 input elements are also serialized.

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));

alert(recursiveEncoded);
alert(recursiveDecoded);
```

The values of `recursiveEncoded` and `recursiveDecoded` are alerted 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);

alert(shallowEncoded);
alert(shallowDecoded);
```

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

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

Example
Serialize a key/value object.

```
var params = { width:1680, height:1050 };
  var str = jQuery.param(params);
  $("#results").text(str);
```

Example
Serialize a few complex objects

```
// <=1.3.2:
$.param({ a: [2,3,4] }) // "a=2&a=3&a=4"
// >=1.4:
$.param({ a: [2,3,4] }) // "a[]=2&a[]=3&a[]=4"

// <=1.3.2:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a=[object+Object]&d=3&d=4&d=[object+Object]"
// >=1.4:
$.param({ a: { b:1,c:2 }, d: [3,4,{ e:5 }] }) // "a[b]=1&a[c]=2&d[]=3&d[]=4&d[2][e]=5"
```

offset()

Get the current coordinates of the first element in the set of matched elements, relative to the document.

The `.offset()` method allows us to retrieve the current position of an element *relative to the document*. Contrast this with `.position()`, which retrieves the current position *relative to the offset parent*. When positioning a new element on top of an existing one for global manipulation (in particular, for implementing drag-and-drop), `.offset()` is the more useful.

`.offset()` returns an object containing the properties `top` and `left`.

Note: jQuery does not support getting the offset coordinates of hidden elements or accounting for borders, margins, or padding set on the body element.

While it is possible to get the coordinates of elements with `visibility:hidden` set, `display:none` is excluded from the rendering tree and thus has a position that is undefined.

Example
Access the offset of the second paragraph:

```
var p = $("p:last");
var offset = p.offset();
p.html( "left: " + offset.left + ", top: " + offset.top );
```

Example
Click to see the offset.

```
$("*", document.body).click(function (e) {
 var offset = $(this).offset();
 e.stopPropagation();
 $("#result").text(this.tagName + " coords ( " + offset.left + ", " +
                offset.top + " )");
});
```

offset(coordinates)
Set the current coordinates of every element in the set of matched elements, relative to the document.

Arguments
coordinates - An object containing the properties `top` and `left`, which are integers indicating the new top and left coordinates for the elements.

The `.offset()` setter method allows us to reposition an element. The element's position is specified *relative to the document*. If the element's `position` style property is currently `static`, it will be set to `relative` to allow for this repositioning.

Example
Set the offset of the second paragraph:

```
$("p:last").offset({ top: 10, left: 30 });
```

css(propertyName)
Get the value of a style property for the first element in the set of matched elements.

Arguments
propertyName - A CSS property.

The `.css()` method is a convenient way to get a style property from the first matched element, especially in light of the different ways browsers access most of those properties (the `getComputedStyle()` method in standards-based browsers versus the `currentStyle` and `runtimeStyle` properties in Internet Explorer) and the different terms browsers use for certain properties. For example, Internet Explorer's DOM implementation refers to the `float` property as `styleFloat`, while W3C standards-compliant browsers refer to it as `cssFloat`. The `.css()` method accounts for such differences, producing the same result no matter which term we use. For example, an element that is floated left will return the string `left` for each of the following three lines:

- `$('div.left').css('float');`
- `$('div.left').css('cssFloat');`
- `$('div.left').css('styleFloat');`

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css('background-color')` and `.css('backgroundColor')`. Different browsers may return CSS color values that are logically but not textually equal, e.g., #FFF, #ffffff, and rgb(255,255,255).

Shorthand CSS properties (e.g. margin, background, border) are not supported. For example, if you want to retrieve the rendered margin, use: `$(elem).css('marginTop')` and `$(elem).css('marginRight')`, and so on.

Example
To access the background color of a clicked div.

```
$("div").click(function () {
 var color = $(this).css("background-color");
 $("#result").html("That div is <span style='color:" +
           color + ";'>" + color + "</span>.");
});
```

css(propertyName, value)
Set one or more CSS properties for the set of matched elements.

Arguments
propertyName - A CSS property name.
value - A value to set for the property.

As with the `.prop()` method, the `.css()` method makes setting properties of elements quick and easy. This method can take either a property name and value as separate parameters, or a single map of key-value pairs (JavaScript object notation).

Also, jQuery can equally interpret the CSS and DOM formatting of multiple-word properties. For example, jQuery understands and returns the correct value for both `.css({'background-color': '#ffe', 'border-left': '5px solid #ccc'})` and `.css({backgroundColor: '#ffe', borderLeft: '5px solid #ccc'})`. Notice that with the DOM notation, quotation marks around the property names are optional, but with CSS notation they're required due to the hyphen in the name.

When using `.css()` as a setter, jQuery modifies the element's `style` property. For example, `$('#mydiv').css('color', 'green')` is equivalent to `document.getElementById('mydiv').style.color = 'green'`. Setting the value of a style property to an empty string - e.g. `$('#mydiv').css('color', '')` - removes that property from an element if it has already been directly applied, whether in the HTML style attribute, through jQuery's `.css()` method, or through direct DOM manipulation of the `style` property. It does not, however, remove a style that has been applied with a CSS rule in a stylesheet or `<style>` element.

As of jQuery 1.6, `.css()` accepts relative values similar to `.animate()`. Relative values are a string starting with `+=` or `-=` to increment or decrement the current value. For example, if an element's padding-left was 10px, `.css( "padding-left", "+=15" )` would result in a total padding-left of 25px.

As of jQuery 1.4, `.css()` allows us to pass a function as the property value:

```
$('div.example').css('width', function(index) {
 return index * 50;
});
```

This example sets the widths of the matched elements to incrementally larger values.

Note: If nothing is returned in the setter function (ie. `function(index, style){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example
To change the color of any paragraph to red on mouseover event.

```
$("p").mouseover(function () {
  $(this).css("color","red");
 });
```

Example
Increase the width of #box by 200 pixels

```
$("#box").one( "click", function () {
  $( this ).css( "width","+=200" );
 });
```

Example
To highlight a clicked word in the paragraph.

```
var words = $("p:first").text().split(" ");
 var text = words.join("</span> <span>");
 $("p:first").html("<span>" + text + "</span>");
 $("span").click(function () {
  $(this).css("background-color","yellow");
 });
```

Example
To set the color of all paragraphs to red and background to blue:

```
$("p").hover(function () {
  $(this).css({'background-color' : 'yellow', 'font-weight' : 'bolder'});
 }, function () {
  var cssObj = {
   'background-color' : '#ddd',
   'font-weight' : '',
   'color' : 'rgb(0,40,244)'
  }
  $(this).css(cssObj);
 });
```

Example
Increase the size of a div when you click it:

```
$("div").click(function() {
  $(this).css({
   width: function(index, value) {
    return parseFloat(value) * 1.2;
   },
   height: function(index, value) {
    return parseFloat(value) * 1.2;
   }

  });
 });
```

unwrap()
Remove the parents of the set of matched elements from the DOM, leaving the matched elements in their place.

The `.unwrap()` method removes the element's parent. This is effectively the inverse of the `.wrap()` method. The matched elements (and their siblings, if any) replace their parents within the DOM structure.

Example
Wrap/unwrap a div around each of the paragraphs.

```
$("button").toggle(function(){
 $("p").wrap("<div></div>");
}, function(){
 $("p").unwrap();
});
```

detach([selector])
Remove the set of matched elements from the DOM.

Arguments
selector - A selector expression that filters the set of matched elements to be removed.

The `.detach()` method is the same as `.remove()`, except that `.detach()` keeps all jQuery data associated with the removed elements. This method is useful when removed elements are to be reinserted into the DOM at a later time.

Example
Detach all paragraphs from the DOM

```
$("p").click(function(){
   $(this).toggleClass("off");
  });
  var p;
  $("button").click(function(){
   if ( p ) {
    p.appendTo("body");
    p = null;
   } else {
    p = $("p").detach();
   }
  });
```

replaceWith(newContent)
Replace each element in the set of matched elements with the provided new content.

Arguments
newContent - The content to insert. May be an HTML string, DOM element, or jQuery object.

The `.replaceWith()` method removes content from the DOM and inserts new content in its place with a single call. Consider this DOM structure:

```
<div class="container">
 <div class="inner first">Hello</div>
 <div class="inner second">And</div>
 <div class="inner third">Goodbye</div>
</div>
```

The second inner `<div>` could be replaced with the specified HTML:

```
$('div.second').replaceWith('<h2>New heading</h2>');
```

This results in the structure:

```
<div class="container">
 <div class="inner first">Hello</div>
 <h2>New heading</h2>
 <div class="inner third">Goodbye</div>
</div>
```

All inner `<div>` elements could be targeted at once:

```
$('div.inner').replaceWith('<h2>New heading</h2>');
```

This causes all of them to be replaced:

```
<div class="container">
 <h2>New heading</h2>
 <h2>New heading</h2>
 <h2>New heading</h2>
</div>
```

An element could also be selected as the replacement:

```
$('div.third').replaceWith($('.first'));
```

This results in the DOM structure:

```
<div class="container">
 <div class="inner second">And</div>
 <div class="inner first">Hello</div>
</div>
```

This example demonstrates that the selected element replaces the target by being moved from its old location, not by being cloned.

The `.replaceWith()` method, like most jQuery methods, returns the jQuery object so that other methods can be chained onto it. However, it must be noted that the *original* jQuery object is returned. This object refers to the element that has been removed from the DOM, not the new element that has replaced it.

As of jQuery 1.4, `.replaceWith()` can also work on disconnected DOM nodes. For example, with the following code, `.replaceWith()` returns a jQuery set containing only a paragraph.:

```
$("<div/>").replaceWith("<p></p>");
```

The `.replaceWith()` method can also take a function as its argument:

```
$('div.container').replaceWith(function() {
 return $(this).contents();
});
```

This results in `<div class="container">` being replaced by its three child `<div>`s. The return value of the function may be an HTML string, DOM element, or jQuery object.

Example
On click, replace the button with a div containing the same word.

```
$("button").click(function () {
 $(this).replaceWith( "<div>" + $(this).text() + "</div>" );
});
```

Example
Replace all paragraphs with bold words.

```
$("p").replaceWith( "<b>Paragraph. </b>" );
```

Example
On click, replace each paragraph with a div that is already in the DOM and selected with the `$()` function. Notice it doesn't clone the object but rather moves it to replace the paragraph.

```
$("p").click(function () {
 $(this).replaceWith( $("div") );
});
```

Example
On button click, replace the containing div with its child divs and append the class name of the selected element to the paragraph.

```
$('button').bind("click", function() {
 var $container = $("div.container").replaceWith(function() {
  return $(this).contents();
 });

 $("p").append( $container.attr("class") );
});
```

wrapInner(wrappingElement)
Wrap an HTML structure around the content of each element in the set of matched elements.

Arguments

wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the content of the matched elements.

The `.wrapInner()` function can take any string or object that could be passed to the `$()` factory function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. The structure will be wrapped around the content of each of the elements in the set of matched elements.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using `.wrapInner()`, we can insert an HTML structure around the content of each inner `<div>` elements like so:

```
$('.inner').wrapInner('<div class="new" />');
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around the content of each matched element:

```
<div class="container">
 <div class="inner">
  <div class="new">Hello</div>
 </div>
 <div class="inner">
  <div class="new">Goodbye</div>
 </div>
</div>
```

The second version of this method allows us to instead specify a callback function. This callback function will be called once for every matched element; it should return a DOM element, jQuery object, or HTML snippet in which to wrap the content of the corresponding element. For example:

```
$('.inner').wrapInner(function() {
 return '<div class="' + this.nodeValue + '" />';
});
```

This will cause each `<div>` to have a class corresponding to the text it wraps:

```
<div class="container">
 <div class="inner">
  <div class="Hello">Hello</div>
 </div>
 <div class="inner">
  <div class="Goodbye">Goodbye</div>
 </div>
</div>
```

Note: When passing a selector string to the `.wrapInner()` function, the expected input is well formed HTML with correctly closed tags. Examples of valid input include:

```
$(elem).wrapInner("<div class='test' />");
$(elem).wrapInner("<div class='test'></div>");
$(elem).wrapInner("<div class="test"></div>");
```

Example

Selects all paragraphs and wraps a bold tag around each of its contents.

```
$("p").wrapInner("<b></b>");
```

Example
Wraps a newly created tree of objects around the inside of the body.

```
$("body").wrapInner("<div><div><p><em><b></b></em></p></div></div>");
```

Example
Selects all paragraphs and wraps a bold tag around each of its contents.

```
$("p").wrapInner(document.createElement("b"));
```

Example
Selects all paragraphs and wraps a jQuery object around each of its contents.

```
$("p").wrapInner($("<span class='red'></span>"));
```

wrapAll(wrappingElement)
Wrap an HTML structure around all elements in the set of matched elements.

Arguments
wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the matched elements.

The `.wrapAll()` function can take any string or object that could be passed to the `$()` function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. The structure will be wrapped around all of the elements in the set of matched elements, as a single group.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using `.wrapAll()`, we can insert an HTML structure around the inner `<div>` elements like so:

```
$('.inner').wrapAll('<div class="new" />');
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around all matched elements:

```
<div class="container">
 <div class="new">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
 </div>
</div>
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrapAll("<div></div>");
```

Example
Wraps a newly created tree of objects around the spans. Notice anything in between the spans gets left out like the (red text) in this example. Even the white space between spans is left out. Click View Source to see the original html.

```
$("span").wrapAll("<div><div><p><em><b></b></em></p></div></div>");
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrapAll(document.createElement("div"));
```

Example
Wrap a jQuery object double depth div around all of the paragraphs. Notice it doesn't move the object but just clones it to wrap around its target.

```
$("p").wrapAll($(".doublediv"));
```

wrap(wrappingElement)
Wrap an HTML structure around each element in the set of matched elements.

Arguments
wrappingElement - An HTML snippet, selector expression, jQuery object, or DOM element specifying the structure to wrap around the matched elements.

The `.wrap()` function can take any string or object that could be passed to the `$()` factory function to specify a DOM structure. This structure may be nested several levels deep, but should contain only one inmost element. A copy of this structure will be wrapped around each of the elements in the set of matched elements. This method returns the original set of elements for chaining purposes.

Consider the following HTML:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Using `.wrap()`, we can insert an HTML structure around the inner `<div>` elements like so:

```
$('.inner').wrap('<div class="new" />');
```

The new `<div>` element is created on the fly and added to the DOM. The result is a new `<div>` wrapped around each matched element:

```
<div class="container">
 <div class="new">
  <div class="inner">Hello</div>
 </div>
 <div class="new">
  <div class="inner">Goodbye</div>
 </div>
</div>
```

The second version of this method allows us to instead specify a callback function. This callback function will be called once for every matched element; it should return a DOM element, jQuery object, or HTML snippet in which to wrap the corresponding element. For example:

```
$('.inner').wrap(function() {
 return '<div class="' + $(this).text() + '" />';
});
```

This will cause each `<div>` to have a class corresponding to the text it wraps:

```
<div class="container">
 <div class="Hello">
  <div class="inner">Hello</div>
 </div>
 <div class="Goodbye">
  <div class="inner">Goodbye</div>
 </div>
</div>
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrap("<div></div>");
```

Example

Wraps a newly created tree of objects around the spans. Notice anything in between the spans gets left out like the (red text) in this example. Even the white space between spans is left out. Click View Source to see the original html.

```
$("span").wrap("<div><div><p><em><b></b></em></p></div></div>");
```

Example
Wrap a new div around all of the paragraphs.

```
$("p").wrap(document.createElement("div"));
```

Example
Wrap a jQuery object double depth div around all of the paragraphs. Notice it doesn't move the object but just clones it to wrap around its target.

```
$("p").wrap($(".doublediv"));
```

before(content, [content])
Insert content, specified by the parameter, before each element in the set of matched elements.

Arguments
content - HTML string, DOM element, or jQuery object to insert before each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert before each element in the set of matched elements.

The `.before()` and `.insertBefore()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.before()`, the selector expression preceding the method is the container before which the content is inserted. With `.insertBefore()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted before the target container.

Consider the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it before several elements at once:

```
$('.inner').before('<p>Test</p>');
```

Each inner `<div>` element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <p>Test</p>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
</div>
```

You can also select an element on the page and insert it before another:

```
$('.container').before($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved before the target (not cloned):

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

In jQuery 1.4, `.before()` and `.after()` will also work on disconnected DOM nodes:

```
$("<div/>").before("<p></p>");
```

The result is a jQuery set that contains a paragraph and a div (in that order).

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.after()`, `.before()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` before the first paragraph:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('p').first().before($newdiv1, [newdiv2, existingdiv1]);
```

Since `.before()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('p').first().before($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Inserts some HTML before all paragraphs.

```
$("p").before("<b>Hello</b>");
```

Example
Inserts a DOM element before all paragraphs.

```
$("p").before( document.createTextNode("Hello") );
```

Example
Inserts a jQuery object (similar to an Array of DOM Elements) before all paragraphs.

```
$("p").before( $("b") );
```

after(content, [content])
Insert content, specified by the parameter, after each element in the set of matched elements.

Arguments
content - HTML string, DOM element, or jQuery object to insert after each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert after each element in the set of matched elements.

The `.after()` and `.insertAfter()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.after()`, the selector expression preceding the method is the container after which the content is inserted. With `.insertAfter()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted after the target container.

Using the following HTML:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

Content can be created and then inserted after several elements at once:

```
$('.inner').after('<p>Test</p>');
```

Each inner element gets this new content:

```
<div class="container">
 <h2>Greetings</h2>
 <div class="inner">Hello</div>
 <p>Test</p>
 <div class="inner">Goodbye</div>
 <p>Test</p>
</div>
```

An element in the DOM can also be selected and inserted after another element:

```
$('.container').after($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved rather than cloned:

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
<h2>Greetings</h2>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Inserting Disconnected DOM nodes

As of jQuery 1.4, .before() and .after() will also work on disconnected DOM nodes. For example, given the following code:

```
$('<div/>').after('<p></p>');
```

The result is a jQuery set containing a div and a paragraph, in that order. That set can be further manipulated, even before it is inserted in the document.

```
$('<div/>').after('<p></p>').addClass('foo')
 .filter('p').attr('id', 'bar').html('hello')
.end()
.appendTo('body');
```

This results in the following elements inserted just before the closing </body> tag:

```
<div class="foo"></div>
<p class="foo" id="bar">hello</p>
```

Passing a Function

As of jQuery 1.4, .after() supports passing a function that returns the elements to insert.

```
$('p').after(function() {
 return '<div>' + this.className + '</div>';
});
```

This example inserts a after each paragraph, with each new containing the class name(s) of its preceding paragraph.

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.before()`, `.after()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` after the first paragraph:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('p').first().after($newdiv1, [newdiv2, existingdiv1]);
```

Since `.after()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('p').first().after($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on the elements that are collected in the code.

Example
Inserts some HTML after all paragraphs.

```
$("p").after("<b>Hello</b>");
```

Example
Inserts a DOM element after all paragraphs.

```
$("p").after( document.createTextNode("Hello") );
```

Example
Inserts a jQuery object (similar to an Array of DOM Elements) after all paragraphs.

```
$("p").after( $("b") );
```

prepend(content, [content])
Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

Arguments
content - DOM element, array of elements, HTML string, or jQuery object to insert at the beginning of each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert at the beginning of each element in the set of matched elements.

The `.prepend()` method inserts the specified content as the first child of each element in the jQuery collection (To insert it as the *last* child, use .append()).

The `.prepend()` and `.prependTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.prepend()`, the selector expression preceding the method is the container into which the content is inserted. With `.prependTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it into several elements at once:

```
$('.inner').prepend('<p>Test</p>');
```

Each `<div class="inner">` element gets this new content:

```
<h2>Greetings</h2>
```

```
<div class="container">
 <div class="inner">
  <p>Test</p>
  Hello
 </div>
 <div class="inner">
  <p>Test</p>
  Goodbye
 </div>
</div>
```

You can also select an element on the page and insert it into another:

```
$('.container').prepend($('h2'));
```

If *a single element* selected this way is inserted elsewhere, it will be moved into the target (*not cloned*):

```
<div class="container">
  <h2>Greetings</h2>
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
```

Important: If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Additional Arguments

Similar to other content-adding methods such as `.append()` and `.before()`, `.prepend()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new `<div>`s and an existing `<div>` as the first three child nodes of the body:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('body').prepend($newdiv1, [newdiv2, existingdiv1]);
```

Since `.prepend()` can accept any number of additional arguments, the same result can be achieved by passing in the three `<div>`s as three separate arguments, like so: `$('body').prepend($newdiv1, newdiv2, existingdiv1)`. The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Prepends some HTML to all paragraphs.

```
$("p").prepend("<b>Hello </b>");
```

Example
Prepends a DOM Element to all paragraphs.

```
$("p").prepend(document.createTextNode("Hello "));
```

Example
Prepends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
$("p").prepend( $("b") );
```

append(content, [content])
Insert content, specified by the parameter, to the end of each element in the set of matched elements.

Arguments

content - DOM element, HTML string, or jQuery object to insert at the end of each element in the set of matched elements.
content - One or more additional DOM elements, arrays of elements, HTML strings, or jQuery objects to insert at the end of each element in the set of matched elements.

The `.append()` method inserts the specified content as the last child of each element in the jQuery collection (To insert it as the *first* child, use .prepend()).

The `.append()` and `.appendTo()` methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With `.append()`, the selector expression preceding the method is the container into which the content is inserted. With `.appendTo()`, on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container.

Consider the following HTML:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
</div>
```

You can create content and insert it into several elements at once:

```
$('.inner').append('<p>Test</p>');
```

Each inner `<div>` element gets this new content:

```
<h2>Greetings</h2>
<div class="container">
 <div class="inner">
  Hello
  <p>Test</p>
 </div>
 <div class="inner">
  Goodbye
  <p>Test</p>
 </div>
</div>
```

You can also select an element on the page and insert it into another:

```
$('.container').append($('h2'));
```

If an element selected this way is inserted elsewhere, it will be moved into the target (not cloned):

```
<div class="container">
 <div class="inner">Hello</div>
 <div class="inner">Goodbye</div>
 <h2>Greetings</h2>
</div>
```

If there is more than one target element, however, cloned copies of the inserted element will be created for each target after the first.

Additional Arguments

Similar to other content-adding methods such as `.prepend()` and `.before()`, `.append()` also supports passing in multiple arguments as input. Supported input includes DOM elements, jQuery objects, HTML strings, and arrays of DOM elements.

For example, the following will insert two new s and an existing as the last three child nodes of the body:

```
var $newdiv1 = $('<div id="object1"/>'),
  newdiv2 = document.createElement('div'),
  existingdiv1 = document.getElementById('foo');

$('body').append($newdiv1, [newdiv2, existingdiv1]);
```

Since .append() can accept any number of additional arguments, the same result can be achieved by passing in the three s as three separate arguments, like so: $('body').append($newdiv1, newdiv2, existingdiv1). The type and number of arguments will largely depend on how you collect the elements in your code.

Example
Appends some HTML to all paragraphs.

```
$("p").append("<strong>Hello</strong>");
```

Example
Appends an Element to all paragraphs.

```
$("p").append(document.createTextNode("Hello"));
```

Example
Appends a jQuery object (similar to an Array of DOM Elements) to all paragraphs.

```
$("p").append( $("strong") );
```

val()
Get the current value of the first element in the set of matched elements.

The .val() method is primarily used to get the values of form elements such as input, select and textarea. In the case of <select multiple="multiple"> elements, the .val() method returns an array containing each selected option; if no option is selected, it returns null.

For selects and checkboxes, you can also use the :selected and :checked selectors to get at values, for example:

```
$('select.foo option:selected').val();  // get the value from a dropdown select
$('select.foo').val();          // get the value from a dropdown select even easier
$('input:checkbox:checked').val();    // get the value from a checked checkbox
$('input:radio[name=bar]:checked').val(); // get the value from a set of radio buttons
```

Note: At present, using .val() on textarea elements strips carriage return characters from the browser-reported value. When this value is sent to the server via XHR however, carriage returns are preserved (or added by browsers which do not include them in the raw value). A workaround for this issue can be achieved using a valHook as follows:

```
$.valHooks.textarea = {
    get: function( elem ) {
        return elem.value.replace( /r?n/g, "rn" );
    }
};
```

Example
Get the single value from a single select and an array of values from a multiple select and display their values.

```
function displayVals() {
   var singleValues = $("#single").val();
   var multipleValues = $("#multiple").val() || [];
   $("p").html("<b>Single:</b> " +
         singleValues +
```

```
        " <b>Multiple:</b> " +
        multipleValues.join(", "));
  }

  $("select").change(displayVals);
  displayVals();
```

Example
Find the value of an input box.

```
$("input").keyup(function () {
   var value = $(this).val();
   $("p").text(value);
  }).keyup();
```

val(value)
Set the value of each element in the set of matched elements.

Arguments
value - A string of text or an array of strings corresponding to the value of each matched element to set as selected/checked.

This method is typically used to set the values of form fields.

Passing an array of element values allows matching `<input type="checkbox">`, `<input type="radio">` and `<option>`s inside of n `<select multiple="multiple">` to be selected. In the case of `<input type="radio">`s that are part of a radio group and `<select multiple="multiple">` the other elements will be deselected.

The `.val()` method allows us to set the value by passing in a function. As of jQuery 1.4, the function is passed two arguments, the current element's index and its current value:

```
$('input:text.items').val(function( index, value ) {
 return value + ' ' + this.className;
});
```

This example appends the string " items" to the text inputs' values.

Example
Set the value of an input box.

```
$("button").click(function () {
   var text = $(this).text();
   $("input").val(text);
  });
```

Example
Use the function argument to modify the value of an input box.

```
$('input').bind('blur', function() {
  $(this).val(function( i, val ) {
   return val.toUpperCase();
  });
 });
```

Example
Set a single select, a multiple select, checkboxes and a radio button .

```
$("#single").val("Single2");
  $("#multiple").val(["Multiple2", "Multiple3"]);
  $("input").val(["check1","check2", "radio1" ]);
```

text()

Get the combined text contents of each element in the set of matched elements, including their descendants.

Unlike the `.html()` method, `.text()` can be used in both XML and HTML documents. The result of the `.text()` method is a string containing the combined text of all matched elements. (Due to variations in the HTML parsers in different browsers, the text returned may vary in newlines and other white space.) Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
 <ul>
 <li>list item 1</li>
 <li>list <strong>item</strong> 2</li>
 </ul>
 </div>
```

The code `$('div.demo-container').text()` would produce the following result:

```
Demonstration Box list item 1 list item 2
```

The `.text()` method cannot be used on form inputs or scripts. To set or get the text value of `input` or `textarea` elements, use the .val() method. To get the value of a script element, use the .html() method.

As of jQuery 1.4, the `.text()` method returns the value of text and CDATA nodes as well as element nodes.

Example

Find the text in the first paragraph (stripping out the html), then set the html of the last paragraph to show it is just text (the red bold is gone).

```
var str = $("p:first").text();
  $("p:last").html(str);
```

text(textString)

Set the content of each element in the set of matched elements to the specified text.

Arguments

textString - A string of text to set as the content of each matched element.

Unlike the `.html()` method, `.text()` can be used in both XML and HTML documents.

We need to be aware that this method escapes the string provided as necessary so that it will render correctly in HTML. To do so, it calls the DOM method `.createTextNode()`, which replaces special characters with their HTML entity equivalents (such as `<` for <). Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
 <ul>
  <li>list item 1</li>
  <li>list <strong>item</strong> 2</li>
 </ul>
</div>
```

The code `$('div.demo-container').text('<p>This is a test.</p>');` will produce the following DOM output:

```
<div class="demo-container">
&lt;p&gt;This is a test.&lt;/p&gt;
</div>
```

It will appear on a rendered page as though the tags were exposed, like this:

```
<p>This is a test</p>
```

The `.text()` method cannot be used on input elements. For input field text, use the .val() method.

As of jQuery 1.4, the `.text()` method allows us to set the text content by passing in a function.

```
$('ul li').text(function(index) {
 return 'item number ' + (index + 1);
});
```

Given an unordered list with three `<li>` elements, this example will produce the following DOM output:

```
<ul>
 <li>item number 1</li>
 <li>item number 2</li>
 <li>item number 3</li>
</ul>
```

Example
Add text to the paragraph (notice the bold tag is escaped).

```
$("p").text("<b>Some</b> new text.");
```

html()
Get the HTML contents of the first element in the set of matched elements.

This method is not available on XML documents.

In an HTML document, `.html()` can be used to get the contents of any element. If the selector expression matches more than one element, only the first match will have its HTML content returned. Consider this code:

```
$('div.demo-container').html();
```

In order for the following `<div>`'s content to be retrieved, it would have to be the first one with `class="demo-container"` in the document:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
</div>
```

The result would look like this:

```
<div class="demo-box">Demonstration Box</div>
```

This method uses the browser's `innerHTML` property. Some browsers may not return HTML that exactly replicates the HTML source in an original document. For example, Internet Explorer sometimes leaves off the quotes around attribute values if they contain only alphanumeric characters.

Example
Click a paragraph to convert it from html to text.

```
$("p").click(function () {
   var htmlStr = $(this).html();
   $(this).text(htmlStr);
  });
```

html(htmlString)
Set the HTML contents of each element in the set of matched elements.

Arguments
htmlString - A string of HTML to set as the content of each matched element.

The `.html()` method is not available in XML documents.

When `.html()` is used to set an element's content, any content that was in that element is completely replaced by the new content. Consider the following HTML:

```
<div class="demo-container">
 <div class="demo-box">Demonstration Box</div>
</div>
```

The content of `<div class="demo-container">` can be set like this:

```
$('div.demo-container')
 .html('<p>All new content. <em>You bet!</em></p>');
```

That line of code will replace everything inside `<div class="demo-container">`:

```
<div class="demo-container">
 <p>All new content. <em>You bet!</em></p>
</div>
```

As of jQuery 1.4, the `.html()` method allows the HTML content to be set by passing in a function.

```
$('div.demo-container').html(function() {
 var emph = '<em>' + $('p').length + ' paragraphs!</em>';
 return '<p>All new content for ' + emph + '</p>';
});
```

Given a document with six paragraphs, this example will set the HTML of `<div class="demo-container">` to `<p>All new content for <em>6 paragraphs!</em></p>`.

This method uses the browser's `innerHTML` property. Some browsers may not generate a DOM that exactly replicates the HTML source provided. For example, Internet Explorer prior to version 8 will convert all `href` properties on links to absolute URLs, and Internet Explorer prior to version 9 will not correctly handle HTML5 elements without the addition of a separate compatibility layer.

Note: In Internet Explorer up to and including version 9, setting the text content of an HTML element may corrupt the text nodes of its children that are being removed from the document as a result of the operation. If you are keeping references to these DOM elements and need them to be unchanged, use `.empty().html(string)` instead of `.html(string)` so that the elements are removed from the document before the new string is assigned to the element.

Example
Add some html to each div.

```
$("div").html("<span class='red'>Hello <b>Again</b></span>");
```

Example
Add some html to each div then immediately do further manipulations to the inserted html.

```
$("div").html("<b>Wow!</b> Such excitement...");
  $("div b").append(document.createTextNode("!!!"))
       .css("color", "red");
```

filter(selector)
Reduce the set of matched elements to those that match the selector or pass the function's test.

Arguments
selector - A string containing a selector expression to match the current set of elements against.

Given a jQuery object that represents a set of DOM elements, the `.filter()` method constructs a new jQuery object from a subset of the matching elements. The supplied selector is tested against each element; all elements matching the selector will be included in the result.

Consider a page with a simple list on it: list item 1 list item 2 list item 3 list item 4 list item 5 list item 6

We can apply this method to the set of list items:

```
$('li').filter(':even').css('background-color', 'red');
```

The result of this call is a red background for items 1, 3, and 5, as they match the selector (recall that `:even` and `:odd` use 0-based indexing).

Using a Filter Function

The second form of this method allows us to filter elements against a function rather than a selector. For each element, if the function returns `true` (or a "truthy" value), the element will be included in the filtered set; otherwise, it will be excluded. Suppose we have a somewhat more involved HTML snippet:

```
<ul>
 <li><strong>list</strong> item 1 -
  one strong tag</li>
 <li><strong>list</strong> item <strong>2</strong> -
  two <span>strong tags</span></li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
 <li>list item 6</li>
</ul>
```

We can select the list items, then filter them based on their contents:

```
$('li').filter(function(index) {
 return $('strong', this).length == 1;
}).css('background-color', 'red');
```

This code will alter the first list item only, as it contains exactly one `<strong>` tag. Within the filter function, `this` refers to each DOM element in turn. The parameter passed to the function tells us the index of that DOM element within the set matched by the jQuery object.

We can also take advantage of the `index` passed through the function, which indicates the 0-based position of the element within the unfiltered set of matched elements:

```
$('li').filter(function(index) {
 return index % 3 == 2;
}).css('background-color', 'red');
```

This alteration to the code will cause the third and sixth list items to be highlighted, as it uses the modulus operator (`%`) to select every item with an `index` value that, when divided by 3, has a remainder of 2.

Example

Change the color of all divs; then add a border to those with a "middle" class.

```
$("div").css("background", "#c8ebcc")
      .filter(".middle")
      .css("border-color", "red");
```

Example

Change the color of all divs; then add a border to the second one (index == 1) and the div with an id of "fourth."

```
$("div").css("background", "#b4b0da")
      .filter(function (index) {
         return index == 1 || $(this).attr("id") == "fourth";
        })
```

```
      .css("border", "3px double red");
```

Example
Select all divs and filter the selection with a DOM element, keeping only the one with an id of "unique".

```
$("div").filter( document.getElementById("unique") )
```

Example
Select all divs and filter the selection with a jQuery object, keeping only the one with an id of "unique".

```
$("div").filter( $("#unique") )
```

toggleClass(className)
Add or remove one or more classes from each element in the set of matched elements, depending on either the class's presence or the value of the switch argument.

Arguments
className - One or more class names (separated by spaces) to be toggled for each element in the matched set.

This method takes one or more class names as its parameter. In the first version, if an element in the matched set of elements already has the class, then it is removed; if an element does not have the class, then it is added. For example, we can apply `.toggleClass()` to a simple `<div>`:

```
<div class="tumble">Some text.</div>
```

The first time we apply `$('div.tumble').toggleClass('bounce')`, we get the following:

```
<div class="tumble bounce">Some text.</div>
```

The second time we apply `$('div.tumble').toggleClass('bounce')`, the `<div>` class is returned to the single `tumble` value:

```
<div class="tumble">Some text.</div>
```

Applying `.toggleClass('bounce spin')` to the same `<div>` alternates between `<div class="tumble bounce spin">` and `<div class="tumble">`.

The second version of `.toggleClass()` uses the second parameter for determining whether the class should be added or removed. If this parameter's value is `true`, then the class is added; if `false`, the class is removed. In essence, the statement:

```
$('#foo').toggleClass(className, addOrRemove);
```

is equivalent to:

```
if (addOrRemove) {
  $('#foo').addClass(className);
 }
 else {
  $('#foo').removeClass(className);
 }
```

As of jQuery 1.4, if no arguments are passed to `.toggleClass()`, all class names on the element the first time `.toggleClass()` is called will be toggled. Also as of jQuery 1.4, the class name to be toggled can be determined by passing in a function.

```
$('div.foo').toggleClass(function() {
 if ($(this).parent().is('.bar')) {
  return 'happy';
 } else {
  return 'sad';
 }
});
```

This example will toggle the `happy` class for `<div class="foo">` elements if their parent element has a class of `bar`; otherwise, it will toggle the `sad` class.

Example
Toggle the class 'highlight' when a paragraph is clicked.

```
$("p").click(function () {
   $(this).toggleClass("highlight");
  });
```

Example
Add the "highlight" class to the clicked paragraph on every third click of that paragraph, remove it every first and second click.

```
var count = 0;
$("p").each(function() {
 var $thisParagraph = $(this);
 var count = 0;
 $thisParagraph.click(function() {
  count++;
  $thisParagraph.find("span").text('clicks: ' + count);
  $thisParagraph.toggleClass("highlight", count % 3 == 0);
 });
});
```

Example
Toggle the class name(s) indicated on the buttons for each div.

```
var cls = ['', 'a', 'a b', 'a b c'];
var divs = $('div.wrap').children();
var appendClass = function() {
 divs.append(function() {
  return '<div>' + (this.className || 'none') + '</div>';
 });
};

appendClass();

$('button').bind('click', function() {
 var tc = this.className || undefined;
 divs.toggleClass(tc);
 appendClass();
});

$('a').bind('click', function(event) {
 event.preventDefault();
 divs.empty().each(function(i) {
  this.className = cls[i];
 });
 appendClass();
});
```

removeClass([className])
Remove a single class, multiple classes, or all classes from each element in the set of matched elements.

Arguments
className - One or more space-separated classes to be removed from the class attribute of each matched element.

If a class name is included as a parameter, then only that class will be removed from the set of matched elements. If no class names are specified in the parameter, all classes will be removed.

More than one class may be removed at a time, separated by a space, from the set of matched elements, like so:

```
$('p').removeClass('myClass yourClass')
```

This method is often used with `.addClass()` to switch elements' classes from one to another, like so:

```
$('p').removeClass('myClass noClass').addClass('yourClass');
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

To replace all existing classes with another class, we can use `.attr('class', 'newClass')` instead.

As of jQuery 1.4, the `.removeClass()` method allows us to indicate the class to be removed by passing in a function.

```
$('li:last').removeClass(function() {
     return $(this).prev().attr('class');
    });
```

This example removes the class name of the penultimate `<li>` from the last `<li>`.

Example
Remove the class 'blue' from the matched elements.

```
$("p:even").removeClass("blue");
```

Example
Remove the class 'blue' and 'under' from the matched elements.

```
$("p:odd").removeClass("blue under");
```

Example
Remove all the classes from the matched elements.

```
$("p:eq(1)").removeClass();
```

removeAttr(attributeName)
Remove an attribute from each element in the set of matched elements.

Arguments
attributeName - An attribute to remove; as of version 1.7, it can be a space-separated list of attributes.

The `.removeAttr()` method uses the JavaScript `removeAttribute()` function, but it has the advantage of being able to be called directly on a jQuery object and it accounts for different attribute naming across browsers.

Note: Removing an inline `onclick` event handler using `.removeAttr()` doesn't achieve the desired effect in Internet Explorer 6, 7, or 8. To avoid potential problems, use `.prop()` instead:

```
$element.prop("onclick", null);
console.log("onclick property: ", $element[0].onclick);
```

Example
Clicking the button enables the input next to it.

```
(function() {
 var inputTitle = $("input").attr("title");
 $("button").click(function () {
  var input = $(this).next();

  if ( input.attr("title") == inputTitle ) {
   input.removeAttr("title")
```

```
  } else {
   input.attr("title", inputTitle);
  }

  $("#log").html( "input title is now " + input.attr("title") );
 });
})();
```

addClass(className)
Adds the specified class(es) to each of the set of matched elements.

Arguments
className - One or more class names to be added to the class attribute of each matched element.

It's important to note that this method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.

More than one class may be added at a time, separated by a space, to the set of matched elements, like so:

```
$("p").addClass("myClass yourClass");
```

This method is often used with `.removeClass()` to switch elements' classes from one to another, like so:

```
$("p").removeClass("myClass noClass").addClass("yourClass");
```

Here, the `myClass` and `noClass` classes are removed from all paragraphs, while `yourClass` is added.

As of jQuery 1.4, the `.addClass()` method's argument can receive a function.

```
$("ul li:last").addClass(function(index) {
 return "item-" + index;
});
```

Given an unordered list with five `<li>` elements, this example adds the class "item-4" to the last `<li>`.

Example
Adds the class "selected" to the matched elements.

```
$("p:last").addClass("selected");
```

Example
Adds the classes "selected" and "highlight" to the matched elements.

```
$("p:last").addClass("selected highlight");
```

Example
Pass in a function to `.addClass()` to add the "green" class to a div that already has a "red" class.

```
$("div").addClass(function(index, currentClass) {
  var addedClass;

  if ( currentClass === "red" ) {
   addedClass = "green";
   $("p").text("There is one green div");
  }

  return addedClass;
 });
```

Version 1.4.1

jQuery.error(message)
Takes a string and throws an exception containing it.

Arguments
message - The message to send out.

This method exists primarily for plugin developers who wish to override it and provide a better display (or more information) for the error messages.

Example
Override jQuery.error for display in Firebug.

```
jQuery.error = console.error;
```

jQuery.parseJSON(json)
Takes a well-formed JSON string and returns the resulting JavaScript object.

Arguments
json - The JSON string to parse.

Passing in a malformed JSON string may result in an exception being thrown. For example, the following are all malformed JSON strings:
- `{test: 1}` (test does not have double quotes around it).
- `{'test': 1}` ('test' is using single quotes instead of double quotes).

Additionally if you pass in nothing, an empty string, null, or undefined, 'null' will be returned from parseJSON. Where the browser provides a native implementation of `JSON.parse`, jQuery uses it to parse the string. For details on the JSON format, see http://json.org/.

Example
Parse a JSON string.

```
var obj = jQuery.parseJSON('{"name":"John"}');
alert( obj.name === "John" );
```

die()
Remove all event handlers previously attached using `.live()` from the elements.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to calling `.unbind()` with no arguments, which is used to remove all handlers attached with `.bind()`.See the discussions of `.live()` and `.unbind()` for further details.

As of jQuery 1.7, use of `.die()` (and its complementary method, `.live()`) is not recommended. Instead, use .off() to remove event handlers bound with .on()

Note: In order for .die() to function correctly, the selector used with it must match exactly the selector initially used with .live().

die(eventType, [handler])
Remove an event handler previously attached using `.live()` from the elements.

Arguments
eventType - A string containing a JavaScript event type, such as `click` or `keydown`.
handler - The function that is no longer to be executed.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to `.unbind()`, which is used to remove handlers attached with `.bind()`.See the discussions of `.live()` and `.unbind()` for further details.

Note: In order for `.die()` to function correctly, the selector used with it must match exactly the selector initially used with `.live()`.

Example
Can bind and unbind events to the colored button.

```
function aClick() {
```

```
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("#theone").live("click", aClick)
       .text("Can Click!");
});
$("#unbind").click(function () {
 $("#theone").die("click", aClick)
       .text("Does nothing...");
});
```

Example
To unbind all live events from all paragraphs, write:

```
$("p").die()
```

Example
To unbind all live click events from all paragraphs, write:

```
$("p").die( "click" )
```

Example
To unbind just one previously bound handler, pass the function in as the second argument:

```
var foo = function () {
// code to handle some kind of event
};

$("p").live("click", foo); // ... now foo will be called when paragraphs are clicked ...

$("p").die("click", foo); // ... foo will no longer be called.
```

width()
Get the current computed width for the first element in the set of matched elements.

The difference between `.css(width)` and `.width()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.width()` method is recommended when an element's width needs to be used in a mathematical calculation.

This method is also able to find the width of the window and document.

```
$(window).width();  // returns width of browser viewport
$(document).width(); // returns width of HTML document
```

Note that `.width()` will always return the content width, regardless of the value of the CSS `box-sizing` property.

Example
Show various widths. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showWidth(ele, w) {
 $("div").text("The width for the " + ele +
       " is " + w + "px.");
}
$("#getp").click(function () {
 showWidth("paragraph", $("p").width());
});
$("#getd").click(function () {
 showWidth("document", $(document).width());
});
```

```
$("#getw").click(function () {
 showWidth("window", $(window).width());
});
```

width(value)

Set the CSS width of each element in the set of matched elements.

Arguments

value - An integer representing the number of pixels, or an integer along with an optional unit of measure appended (as a string).

When calling `.width("value")`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, any valid CSS measurement may be used for the width (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS width property does not include padding, border, or margin, unless the `box-sizing` CSS property is used.

If no explicit unit is specified (like "em" or "%") then "px" is assumed.

Note that `.width("value")` sets the width of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the `outerWidth` of the box instead of the content width.

Example

Change the width of each div the first time it is clicked (and change its color).

```
(function() {
 var modWidth = 50;
 $("div").one('click', function () {
  $(this).width(modWidth).addClass("mod");
 modWidth -= 8;
 });
})();
```

height()

Get the current computed height for the first element in the set of matched elements.

The difference between `.css('height')` and `.height()` is that the latter returns a unit-less pixel value (for example, `400`) while the former returns a value with units intact (for example, `400px`). The `.height()` method is recommended when an element's height needs to be used in a mathematical calculation.

This method is also able to find the height of the window and document.

```
$(window).height();  // returns height of browser viewport
$(document).height(); // returns height of HTML document
```

Note that `.height()` will always return the content height, regardless of the value of the CSS `box-sizing` property.

Note: Although `style` and `script` tags will report a value for `.width()` or `height()` when absolutely positioned and given `display:block`, it is strongly discouraged to call those methods on these tags. In addition to being a bad practice, the results may also prove unreliable.

Example

Show various heights. Note the values are from the iframe so might be smaller than you expected. The yellow highlight shows the iframe body.

```
function showHeight(ele, h) {
   $("div").text("The height for the " + ele +
          " is " + h + "px.");
  }
  $("#getp").click(function () {
   showHeight("paragraph", $("p").height());
  });
  $("#getd").click(function () {
```

```
   showHeight("document", $(document).height());
  });
  $("#getw").click(function () {
   showHeight("window", $(window).height());
  });
```

height(value)
Set the CSS height of every matched element.

Arguments
value - An integer representing the number of pixels, or an integer with an optional unit of measure appended (as a string).

When calling `.height(value)`, the value can be either a string (number and unit) or a number. If only a number is provided for the value, jQuery assumes a pixel unit. If a string is provided, however, a valid CSS measurement must be provided for the height (such as `100px`, `50%`, or `auto`). Note that in modern browsers, the CSS height property does not include padding, border, or margin.

If no explicit unit was specified (like 'em' or '%') then "px" is concatenated to the value.

Note that `.height(value)` sets the height of the box in accordance with the CSS `box-sizing` property. Changing this property to `border-box` will cause this function to change the outerHeight of the box instead of the content height.

Example
To set the height of each div on click to 30px plus a color change.

```
$("div").one('click', function () {
   $(this).height(30)
       .css({cursor:"auto", backgroundColor:"green"});
  });
```

Version 1.4.2

undelegate()
Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

The `.undelegate()` method is a way of removing event handlers that have been bound using .delegate(). **As of jQuery 1.7**, the .on() and .off() methods are preferred for attaching and removing event handlers.

Example
Can bind and unbind events to the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("body").delegate("#theone", "click", aClick)
  .find("#theone").text("Can Click!");
});
$("#unbind").click(function () {
 $("body").undelegate("#theone", "click", aClick)
  .find("#theone").text("Does nothing...");
});
```

Example
To unbind all delegated events from all paragraphs, write:

```
$("p").undelegate()
```

Example
To unbind all delegated click events from all paragraphs, write:

```
$("p").undelegate( "click" )
```

Example
To undelegate just one previously bound handler, pass the function in as the third argument:

```
var foo = function () {
 // code to handle some kind of event
};

// ... now foo will be called when paragraphs are clicked ...
$("body").delegate("p", "click", foo);


// ... foo will no longer be called.
$("body").undelegate("p", "click", foo);
```

Example
To unbind all delegated events by their namespace:

```
var foo = function () {
 // code to handle some kind of event
};

// delegate events under the ".whatever" namespace
$("form").delegate(":button", "click.whatever", foo);

$("form").delegate("input[type='text']", "keypress.whatever", foo);

// unbind all events delegated under the ".whatever" namespace

$("form").undelegate(".whatever");
```

delegate(selector, eventType, handler(eventObject))
Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

Arguments
selector - A selector to filter the elements that trigger the event.
eventType - A string containing one or more space-separated JavaScript event types, such as "click" or "keydown," or custom event names.
handler(eventObject) - A function to execute at the time the event is triggered.

As of jQuery 1.7, `.delegate()` has been superseded by the .on() method. For earlier versions, however, it remains the most effective means to use event delegation. More information on event binding and delegation is in the .on() method. In general, these are the equivalent templates for the two methods:

```
$(elements).delegate(selector, events, data, handler); // jQuery 1.4.3+
$(elements).on(events, selector, data, handler);    // jQuery 1.7+
```

For example, the following `.delegate()` code:

```
$("table").delegate("td", "click", function() {
 $(this).toggleClass("chosen");
});
```

is equivalent to the following code written using `.on()`:

```
$("table").on("click", "td", function() {
 $(this).toggleClass("chosen");
});
```

To remove events attached with `delegate()`, see the .undelegate() method.

Passing and handling event data works the same way as it does for `.on()`.

Example
Click a paragraph to add another. Note that .delegate() attaches a click event handler to all paragraphs - even new ones.

```
$("body").delegate("p", "click", function(){
   $(this).after("<p>Another paragraph!</p>");
  });
```

Example
To display each paragraph's text in an alert box whenever it is clicked:

```
$("body").delegate("p", "click", function(){
 alert( $(this).text() );
});
```

Example
To cancel a default action and prevent it from bubbling up, return false:

```
$("body").delegate("a", "click", function() { return false; })
```

Example
To cancel only the default action by using the preventDefault method.

```
$("body").delegate("a", "click", function(event){
 event.preventDefault();
});
```

Example
Can bind custom events too.

```
$("body").delegate("p", "myCustomEvent", function(e, myName, myValue){
   $(this).text("Hi there!");
   $("span").stop().css("opacity", 1)
        .text("myName = " + myName)
        .fadeIn(30).fadeOut(1000);
  });
  $("button").click(function () {
   $("p").trigger("myCustomEvent");
  });
```

Version 1.4.3

jQuery.now()
Return a number representing the current time.

The `$.now()` method is a shorthand for the number returned by the expression `(new Date).getTime()`.

jQuery.cssHooks
Hook directly into jQuery to override how particular CSS properties are retrieved or set, normalize CSS property naming, or create custom properties.

The `$.cssHooks` object provides a way to define functions for getting and setting particular CSS values. It can also be used to create new cssHooks for normalizing CSS3 features such as box shadows and gradients.

For example, some versions of Webkit-based browsers require `-webkit-border-radius` to set the `border-radius` on an element, while earlier Firefox versions require `-moz-border-radius`. A css hook can normalize these vendor-prefixed properties to let `.css()` accept a single, standard property name (`border-radius`, or with DOM property syntax, `borderRadius`).

In addition to providing fine-grained control over how specific style properties are handled, `$.cssHooks` also extends the set of properties available to the `.animate()` method.

Defining a new css hook is straight-forward. The skeleton template below can serve as a guide to creating your own.

```
(function($) {
 // first, check to see if cssHooks are supported
 if ( !$.cssHooks ) {
  // if not, output an error message
  throw("jQuery 1.4.3 or above is required for this plugin to work");
  return;
 }

 $.cssHooks["someCSSProp"] = {
  get: function( elem, computed, extra ) {
   // handle getting the CSS property
  },
  set: function( elem, value ) {
   // handle setting the CSS value
  }
 };
})(jQuery);
```

Feature Testing

Before normalizing a vendor-specific CSS property, first determine whether the browser supports the standard property or a vendor-prefixed variation. For example, to check for support of the `border-radius` property, see if any variation is a member of a temporary element's `style` object.

```
(function($) {
 function styleSupport( prop ) {
  var vendorProp, supportedProp,

    // capitalize first character of the prop to test vendor prefix
    capProp = prop.charAt(0).toUpperCase() + prop.slice(1),
    prefixes = [ "Moz", "Webkit", "O", "ms" ],
    div = document.createElement( "div" );

  if ( prop in div.style ) {

   // browser supports standard CSS property name
   supportedProp = prop;
  } else {

   // otherwise test support for vendor-prefixed property names
   for ( var i = 0; i < prefixes.length; i++ ) {
    vendorProp = prefixes[i] + capProp;
    if ( vendorProp in div.style ) {
     supportedProp = vendorProp;
     break;
    }
   }
  }

  // avoid memory leak in IE
  div = null;

  // add property to $.support so it can be accessed elsewhere
  $.support[ prop ] = supportedProp;

  return supportedProp;
 }
```

```
 // call the function, e.g. testing for "border-radius" support:
 styleSupport( "borderRadius" );
})(jQuery);
```

Defining a complete css hook

To define a complete css hook, combine the support test with a filled-out version of the skeleton template provided in the first example:

```
(function($) {
 if ( !$.cssHooks ) {
  throw("jQuery 1.4.3+ is needed for this plugin to work");
  return;
 }

 function styleSupport( prop ) {
  var vendorProp, supportedProp,
    capProp = prop.charAt(0).toUpperCase() + prop.slice(1),
    prefixes = [ "Moz", "Webkit", "O", "ms" ],
    div = document.createElement( "div" );

  if ( prop in div.style ) {
   supportedProp = prop;
  } else {
   for ( var i = 0; i < prefixes.length; i++ ) {
    vendorProp = prefixes[i] + capProp;
    if ( vendorProp in div.style ) {
     supportedProp = vendorProp;
     break;
    }
   }
  }

  div = null;
  $.support[ prop ] = supportedProp
  return supportedProp;
 }

 var borderRadius = styleSupport( "borderRadius" );

 // Set cssHooks only for browsers that
 // support a vendor-prefixed border radius
 if ( borderRadius && borderRadius !== "borderRadius" ) {
  $.cssHooks.borderRadius = {
   get: function( elem, computed, extra ) {
    return $.css( elem, borderRadius );
   },
   set: function( elem, value) {
    elem.style[ borderRadius ] = value;
   }
  };
 }
})(jQuery);
```

You can then set the border radius in a supported browser using either the DOM (camelCased) style or the CSS (hyphenated) style:

```
$("#element").css("borderRadius", "10px");
$("#another").css("border-radius", "20px");
```

If the browser lacks support for any form of the CSS property, vendor-prefixed or not, the style is not applied to the element. However, if the browser supports a proprietary alternative, it can be applied to the cssHooks instead.

```
(function($) {
// feature test for support of a CSS property
// and a proprietary alternative
// ...

if ( $.support.someCSSProp && $.support.someCSSProp !== "someCSSProp" ) {

 // Set cssHooks for browsers that
 // support only a vendor-prefixed someCSSProp
 $.cssHooks.someCSSProp = {
  get: function( elem, computed, extra ) {
   return $.css( elem, $.support.someCSSProp );
  },
  set: function( elem, value) {
   elem.style[ $.support.someCSSProp ] = value;
  }
 };
} else if ( supportsProprietaryAlternative ) {
 $.cssHooks.someCSSProp = {
  get: function( elem, computed, extra ) {
   // Handle crazy conversion from the proprietary alternative
  },
  set: function( elem, value ) {
   // Handle crazy conversion to the proprietary alternative
  }
 }
}

})(jQuery);
```

Special units

By default, jQuery adds a "px" unit to the values passed to the `.css()` method. This behavior can be prevented by adding the property to the `jQuery.cssNumber` object

```
$.cssNumber["someCSSProp"] = true;
```

Animating with cssHooks

A css hook can also hook into jQuery's animation mechanism by adding a property to the `jQuery.fx.step` object:

```
$.fx.step["someCSSProp"] = function(fx){
 $.cssHooks["someCSSProp"].set( fx.elem, fx.now + fx.unit );
};
```

Note that this works best for simple numeric-value animations. More custom code may be required depending on the CSS property, the type of value it returns, and the animation's complexity.

jQuery.type(obj)

Determine the internal JavaScript [[Class]] of an object.

Arguments

obj - Object to get the internal JavaScript [[Class]] of.

A number of techniques are used to determine the exact return value for an object. The [[Class]] is determined as follows:

- If the object is undefined or null, then "undefined" or "null" is returned accordingly.
- jQuery.type(undefined) === "undefined"
- jQuery.type() === "undefined"
- jQuery.type(window.notDefined) === "undefined"
- jQuery.type(null) === "null"
- If the object has an internal [[Class]] equivalent to one of the browser's built-in objects, the associated name is returned. (More details about this technique.)
- jQuery.type(true) === "boolean"
- jQuery.type(3) === "number"
- jQuery.type("test") === "string"
- jQuery.type(function(){}) === "function"
- jQuery.type([]) === "array"
- jQuery.type(new Date()) === "date"
- jQuery.type(/test/) === "regexp"
- Everything else returns "object" as its type.

Example

Find out if the parameter is a RegExp.

```
$("b").append( "" + jQuery.type(/test/) );
```

jQuery.isWindow(obj)

Determine whether the argument is a window.

Arguments

obj - Object to test whether or not it is a window.

This is used in a number of places in jQuery to determine if we're operating against a browser window (such as the current window or an iframe).

Example

Finds out if the parameter is a window.

```
$("b").append( "" + $.isWindow(window) );
```

jQuery.fx.interval

The rate (in milliseconds) at which animations fire.

This property can be manipulated to adjust the number of frames per second at which animations will run. The default is 13 milliseconds. Making this a lower number could make the animations run smoother in faster browsers (such as Chrome) but there may be performance and CPU implications of doing so.

Since jQuery uses one global interval, no animation should be running or all animations should stop for the change of this property to take effect.

Note: `jQuery.fx.interval` currently has no effect in browsers that support the `requestAnimationFrame` property, such as Google Chrome 11. This behavior is subject to change in a future release.

Example

Cause all animations to run with less frames.

```
jQuery.fx.interval = 100;

$("input").click(function(){
 $("div").toggle( 3000 );
});
```

event.namespace

The namespace specified when the event was triggered.

This will likely be used primarily by plugin authors who wish to handle tasks differently depending on the event namespace used.

Example
Determine the event namespace used.

```
$("p").bind("test.something", function(event) {
 alert( event.namespace );
});
$("button").click(function(event) {
 $("p").trigger("test.something");
});
```

undelegate()
Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

The `.undelegate()` method is a way of removing event handlers that have been bound using .delegate(). **As of jQuery 1.7**, the .on() and .off() methods are preferred for attaching and removing event handlers.

Example
Can bind and unbind events to the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("body").delegate("#theone", "click", aClick)
  .find("#theone").text("Can Click!");
});
$("#unbind").click(function () {
 $("body").undelegate("#theone", "click", aClick)
  .find("#theone").text("Does nothing...");
});
```

Example
To unbind all delegated events from all paragraphs, write:

```
$("p").undelegate()
```

Example
To unbind all delegated click events from all paragraphs, write:

```
$("p").undelegate( "click" )
```

Example
To undelegate just one previously bound handler, pass the function in as the third argument:

```
var foo = function () {
 // code to handle some kind of event
};

// ... now foo will be called when paragraphs are clicked ...
$("body").delegate("p", "click", foo);


// ... foo will no longer be called.
$("body").undelegate("p", "click", foo);
```

Example
To unbind all delegated events by their namespace:

```
var foo = function () {
```

```
 // code to handle some kind of event
};

// delegate events under the ".whatever" namespace
$("form").delegate(":button", "click.whatever", foo);

$("form").delegate("input[type='text']", "keypress.whatever", foo);

// unbind all events delegated under the ".whatever" namespace

$("form").undelegate(".whatever");
```

delegate(selector, eventType, handler(eventObject))

Attach a handler to one or more events for all elements that match the selector, now or in the future, based on a specific set of root elements.

Arguments

selector - A selector to filter the elements that trigger the event.
eventType - A string containing one or more space-separated JavaScript event types, such as "click" or "keydown," or custom event names.
handler(eventObject) - A function to execute at the time the event is triggered.

As of jQuery 1.7, `.delegate()` has been superseded by the .on() method. For earlier versions, however, it remains the most effective means to use event delegation. More information on event binding and delegation is in the .on() method. In general, these are the equivalent templates for the two methods:

```
$(elements).delegate(selector, events, data, handler); // jQuery 1.4.3+
$(elements).on(events, selector, data, handler);    // jQuery 1.7+
```

For example, the following `.delegate()` code:

```
$("table").delegate("td", "click", function() {
 $(this).toggleClass("chosen");
});
```

is equivalent to the following code written using `.on()`:

```
$("table").on("click", "td", function() {
 $(this).toggleClass("chosen");
});
```

To remove events attached with `delegate()`, see the .undelegate() method.

Passing and handling event data works the same way as it does for `.on()`.

Example

Click a paragraph to add another. Note that .delegate() attaches a click event handler to all paragraphs - even new ones.

```
$("body").delegate("p", "click", function(){
   $(this).after("<p>Another paragraph!</p>");
  });
```

Example

To display each paragraph's text in an alert box whenever it is clicked:

```
$("body").delegate("p", "click", function(){
 alert( $(this).text() );
});
```

Example

To cancel a default action and prevent it from bubbling up, return false:

```
$("body").delegate("a", "click", function() { return false; })
```

Example
To cancel only the default action by using the preventDefault method.

```
$("body").delegate("a", "click", function(event){
 event.preventDefault();
});
```

Example
Can bind custom events too.

```
$("body").delegate("p", "myCustomEvent", function(e, myName, myValue){
   $(this).text("Hi there!");
   $("span").stop().css("opacity", 1)
        .text("myName = " + myName)
        .fadeIn(30).fadeOut(1000);
  });
  $("button").click(function () {
   $("p").trigger("myCustomEvent");
  });
```

focusout(handler(eventObject))
Bind an event handler to the "focusout" JavaScript event.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('focusout', handler)`.

The `focusout` event is sent to an element when it, or any element inside of it, loses focus. This is distinct from the blur event in that it supports detecting the loss of focus from parent elements (in other words, it supports event bubbling).

This event will likely be used together with the focusin event.

Example
Watch for a loss of focus to occur inside paragraphs and note the difference between the `focusout` count and the `blur` count.

```
var fo = 0, b = 0;
$("p").focusout(function() {
 fo++;
 $("#fo")
  .text("focusout fired: " + fo + "x");
}).blur(function() {
 b++;
 $("#b")
  .text("blur fired: " + b + "x");

});
```

focusin(handler(eventObject))
Bind an event handler to the "focusin" event.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('focusin', handler)`.

The `focusin` event is sent to an element when it, or any element inside of it, gains focus. This is distinct from the focus event in that it supports detecting the focus event on parent elements (in other words, it supports event bubbling).

This event will likely be used together with the focusout event.

Example
Watch for a focus to occur within the paragraphs on the page.

```
$("p").focusin(function() {
     $(this).find("span").css('display','inline').fadeOut(1000);
  });
```

jQuery.data(element, key, value)
Store arbitrary data associated with the specified element. Returns the value that was set.

Arguments
element - The DOM element to associate with the data.
key - A string naming the piece of data to set.
value - The new data value.

Note: This is a low-level method; a more convenient `.data()` is also available.

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');
```

Note: 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.

Example
Store then retrieve a value from the div element.

```
var div = $("div")[0];
  jQuery.data(div, "test", { first: 16, last: "pizza!" });
  $("span:first").text(jQuery.data(div, "test").first);
  $("span:last").text(jQuery.data(div, "test").last);
```

jQuery.data(element, key)
Returns value at named data store for the element, as set by `jQuery.data(element, name, value)`, or the full data store for the element.

Arguments
element - The DOM element to query for the data.
key - Name of the data stored.

Note: This is a low-level method; a more convenient `.data()` is also available.

Regarding HTML5 data-* attributes: This low-level method does NOT retrieve the `data-*` attributes unless the more convenient `.data()` method has already retrieved them.

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 from memory leaks. We can retrieve several distinct values for a single element one at a time, or as a set:

```
alert(jQuery.data( document.body, 'foo' ));
alert(jQuery.data( document.body ));
```

The above lines alert the data values that were set on the `body` element. If nothing was set on that element, an empty string is returned.

Calling `jQuery.data(element)` retrieves all of the element's associated values as a JavaScript object. Note that jQuery itself uses this method to store data for internal use, such as event handlers, so do not assume that it contains only data that your own code has stored.

Note: 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.

Example
Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value, div = $("div")[0];

 switch ($("button").index(this)) {
  case 0 :
   value = jQuery.data(div, "blah");
   break;
  case 1 :
   jQuery.data(div, "blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   jQuery.data(div, "blah", 86);
   value = "Stored!";
   break;
  case 3 :
   jQuery.removeData(div, "blah");
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

keydown(handler(eventObject))
Bind an event handler to the "keydown" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('keydown', handler)` in the first and second variations, and `.trigger('keydown')` in the third.

The `keydown` event is sent to an element when the user first presses a key on the keyboard. It can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$('#target').keydown(function() {
 alert('Handler for .keydown() called.');
});
```

Now when the insertion point is inside the field, pressing a key displays the alert:

Handler for .keydown() called.

To trigger the event manually, apply `.keydown()` without an argument:

```
$('#other').click(function() {
 $('#target').keydown();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which key was pressed, examine the event object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the key code. This code corresponds to a key on the keyboard, including codes for special keys such as arrows. For catching actual text entry, `.keypress()` may be a better choice.

Example

Show the event object for the keydown handler when a key is pressed in the input.

```
var xTriggered = 0;
$('#target').keydown(function(event) {
 if (event.which == 13) {
   event.preventDefault();
  }
  xTriggered++;
  var msg = 'Handler for .keydown() called ' + xTriggered + ' time(s).';
 $.print(msg, 'html');
 $.print(event);
});

$('#other').click(function() {
 $('#target').keydown();
});
```

data(key, value)

Store arbitrary data associated with the matched elements.

Arguments

key - A string naming the piece of data to set.
value - The new data value; it can be any Javascript type including Array or Object.

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 later:

```
$('body').data('foo', 52);
$('body').data('bar', { myType: 'test', count: 40 });

$('body').data('foo'); // 52
$('body').data(); // {foo: 52, bar: { myType: 'test', count: 40 }}
```

In jQuery 1.4.3 setting an element's data object with `.data(obj)` extends the data previously stored with that element. jQuery itself uses the `.data()` method to save information under the names 'events' and 'handle', and also reserves any data name starting with an underscore ('_') for internal use.

Prior to jQuery 1.4.3 (starting in jQuery 1.4) the .data() method completely replaced all data, instead of just extending the data object. If you are using third-party plugins it may not be advisable to completely replace the element's data object, since plugins may have also set data.

Due to the way browsers interact with plugins and external code, the `.data()` method cannot be used on `<object>` (unless it's a Flash plugin), `<applet>` or `<embed>` elements.

Example

Store then retrieve a value from the div element.

```
$("div").data("test", { first: 16, last: "pizza!" });
$("span:first").text($("div").data("test").first);
$("span:last").text($("div").data("test").last);
```

data(key)

Returns value at named data store for the first element in the jQuery collection, as set by data(name, value).

Arguments

key - Name of the data stored.

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 retrieve several distinct values for a single element one at a time, or as a set:

```
alert($('body').data('foo'));
alert($('body').data());
```

The above lines alert the data values that were set on the `body` element. If no data at all was set on that element, `undefined` is returned.

```
alert( $("body").data("foo")); //undefined
$("body").data("bar", "foobar");
alert( $("body").data("bar")); //foobar
```

HTML5 data-* Attributes

As of jQuery 1.4.3 HTML 5 data- attributes will be automatically pulled in to jQuery's data object. The treatment of attributes with embedded dashes was changed in jQuery 1.6 to conform to the W3C HTML5 specification.

For example, given the following HTML:

```
<div data-role="page" data-last-value="43" data-hidden="true" data-options='{"name":"John"}'></div>
```

All of the following jQuery code will work.

```
$("div").data("role") === "page";
$("div").data("lastValue") === 43;
$("div").data("hidden") === true;
$("div").data("options").name === "John";
```

Every attempt is made to convert the string to a JavaScript value (this includes booleans, numbers, objects, arrays, and null) otherwise it is left as a string. To retrieve the value's attribute as a string without any attempt to convert it, use the `attr()` method. When the data attribute is an object (starts with '{') or array (starts with '[') then `jQuery.parseJSON` is used to parse the string; it must follow valid JSON syntax*including quoted property names*. The data- attributes are pulled in the first time the data property is accessed and then are no longer accessed or mutated (all data values are then stored internally in jQuery).

Calling `.data()` with no parameters retrieves all of the values as a JavaScript object. This object can be safely cached in a variable as long as a new object is not set with `.data(obj)`. Using the object directly to get or set values is faster than making individual calls to `.data()` to get or set each value:

```
var mydata = $("#mydiv").data();
if ( mydata.count < 9 ) {
  mydata.count = 43;
  mydata.status = "embiggened";
}
```

Example
Get the data named "blah" stored at for an element.

```
$("button").click(function(e) {
 var value;

 switch ($("button").index(this)) {
  case 0 :
   value = $("div").data("blah");
   break;
  case 1 :
   $("div").data("blah", "hello");
   value = "Stored!";
   break;
  case 2 :
   $("div").data("blah", 86);
   value = "Stored!";
   break;
  case 3 :
   $("div").removeData("blah");
   value = "Removed!";
   break;
 }

 $("span").text("" + value);
});
```

scroll(handler(eventObject))
Bind an event handler to the "scroll" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('scroll', handler)` in the first and second variations, and `.trigger('scroll')` in the third.

The `scroll` event is sent to an element when the user scrolls to a different place in the element. It applies to `window` objects, but also to scrollable frames and elements with the `overflow` CSS property set to `scroll` (or `auto` when the element's explicit height or width is less than the height or width of its contents).

For example, consider the HTML:

```
<div id="target" style="overflow: scroll; width: 200px; height: 100px;">
 Lorem ipsum dolor sit amet, consectetur adipisicing elit,
 sed do eiusmod tempor incididunt ut labore et dolore magna
 aliqua. Ut enim ad minim veniam, quis nostrud exercitation
 ullamco laboris nisi ut aliquip ex ea commodo consequat.
 Duis aute irure dolor in reprehenderit in voluptate velit
 esse cillum dolore eu fugiat nulla pariatur. Excepteur
 sint occaecat cupidatat non proident, sunt in culpa qui
 officia deserunt mollit anim id est laborum.
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The style definition is present to make the target element small enough to be scrollable:

The `scroll` event handler can be bound to this element:

```
$('#target').scroll(function() {
 $('#log').append('<div>Handler for .scroll() called.</div>');
});
```

Now when the user scrolls the text up or down, one or more messages are appended to `<div id="log"></div>`:

Handler for .scroll() called.

To trigger the event manually, apply `.scroll()` without an argument:

```
$('#other').click(function() {
 $('#target').scroll();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

A `scroll` event is sent whenever the element's scroll position changes, regardless of the cause. A mouse click or drag on the scroll bar, dragging inside the element, pressing the arrow keys, or using the mouse's scroll wheel could cause this event.

Example
To do something when your page is scrolled:

```
$("p").clone().appendTo(document.body);
  $("p").clone().appendTo(document.body);
  $("p").clone().appendTo(document.body);
  $(window).scroll(function () {
   $("span").css("display", "inline").fadeOut("slow");
  });
```

resize(handler(eventObject))
Bind an event handler to the "resize" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('resize', handler)` in the first and second variations, and `.trigger('resize')` in the third.

The `resize` event is sent to the `window` element when the size of the browser window changes:

```
$(window).resize(function() {
 $('#log').append('<div>Handler for .resize() called.</div>');
});
```

Now whenever the browser window's size is changed, the message is appended to one or more times, depending on the browser.

Code in a `resize` handler should never rely on the number of times the handler is called. Depending on implementation, `resize` events can be sent continuously as the resizing is in progress (the typical behavior in Internet Explorer and WebKit-based browsers such as Safari and Chrome), or only once at the end of the resize operation (the typical behavior in some other browsers such as Opera).

Example
To see the window width while (or after) it is resized, try:

```
$(window).resize(function() {
 $('body').prepend('<div>' + $(window).width() + '</div>');
});
```

keyup(handler(eventObject))
Bind an event handler to the "keyup" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('keyup', handler)` in the first two variations, and `.trigger('keyup')` in the third.

The `keyup` event is sent to an element when the user releases a key on the keyboard. It can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$('#target').keyup(function() {
 alert('Handler for .keyup() called.');
});
```

Now when the insertion point is inside the field and a key is pressed and released, the alert is displayed:

Handler for .keyup() called.

To trigger the event manually, apply `.keyup()` without arguments:

```
$('#other').click(function() {
 $('#target').keyup();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which key was pressed, examine the event object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the key code. This code corresponds to a key on the keyboard, including codes for special keys such as arrows. For catching actual text entry, `.keypress()` may be a better choice.

Example

Show the event object for the keyup handler (using a simple $.print plugin) when a key is released in the input.

```
var xTriggered = 0;
$('#target').keyup(function(event) {
  xTriggered++;
  var msg = 'Handler for .keyup() called ' + xTriggered + ' time(s).';
 $.print(msg, 'html');
 $.print(event);
}).keydown(function(event) {
 if (event.which == 13) {
  event.preventDefault();
 }
});

$('#other').click(function() {
 $('#target').keyup();
});
```

keypress(handler(eventObject))

Bind an event handler to the "keypress" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

Note: as the `keypress` event isn't covered by any official specification, the actual behavior encountered when using it may differ across browsers, browser versions, and platforms.

This method is a shortcut for `.bind("keypress", handler)` in the first two variations, and `.trigger("keypress")` in the third.

The `keypress` event is sent to an element when the browser registers keyboard input. This is similar to the `keydown` event, except in the case of key repeats. If the user presses and holds a key, a `keydown` event is triggered once, but separate `keypress` events are triggered for each inserted character. In addition, modifier keys (such as Shift) trigger `keydown` events but not `keypress` events.

A `keypress` event handler can be attached to any element, but the event is only sent to the element that has the focus. Focusable elements can vary between browsers, but form elements can always get focus so are reasonable candidates for this event type.

For example, consider the HTML:

```
<form>
 <fieldset>
  <input id="target" type="text" value="Hello there" />
 </fieldset>
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the input field:

```
$("#target").keypress(function() {
 alert("Handler for .keypress() called.");
});
```

Now when the insertion point is inside the field, pressing a key displays the alert:

Handler for .keypress() called.

The message repeats if the key is held down. To trigger the event manually, apply `.keypress()` without an argument::

```
$('#other').click(function() {
 $("#target").keypress();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If key presses anywhere need to be caught (for example, to implement global shortcut keys on a page), it is useful to attach this behavior to the `document` object. Because of event bubbling, all key presses will make their way up the DOM to the `document` object unless explicitly stopped.

To determine which character was entered, examine the `event` object that is passed to the handler function. While browsers use differing properties to store this information, jQuery normalizes the `.which` property so you can reliably use it to retrieve the character code.

Note that `keydown` and `keyup` provide a code indicating which key is pressed, while `keypress` indicates which character was entered. For example, a lowercase "a" will be reported as 65 by `keydown` and `keyup`, but as 97 by `keypress`. An uppercase "A" is reported as 65 by all events. Because of this distinction, when catching special keystrokes such as arrow keys, `.keydown()` or `.keyup()` is a better choice.

Example

Show the event object when a key is pressed in the input. Note: This demo relies on a simple $.print() plugin (http://api.jquery.com/scripts/events.js) for the event object's output.

```
var xTriggered = 0;
$("#target").keypress(function(event) {
```

```
 if ( event.which == 13 ) {
   event.preventDefault();
  }
  xTriggered++;
  var msg = "Handler for .keypress() called " + xTriggered + " time(s).";
 $.print( msg, "html" );
 $.print( event );
});

$("#other").click(function() {
 $("#target").keypress();
});
```

submit(handler(eventObject))

Bind an event handler to the "submit" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('submit', handler)` in the first variation, and `.trigger('submit')` in the third.

The `submit` event is sent to an element when the user is attempting to submit a form. It can only be attached to `<form>` elements. Forms can be submitted either by clicking an explicit `<input type="submit">`, `<input type="image">`, or `<button type="submit">`, or by pressing Enter when certain form elements have focus.

Depending on the browser, the Enter key may only cause a form submission if the form has exactly one text field, or only when there is a submit button present. The interface should not rely on a particular behavior for this key unless the issue is forced by observing the keypress event for presses of the Enter key.

For example, consider the HTML:

```
<form id="target" action="destination.html">
 <input type="text" value="Hello there" />
 <input type="submit" value="Go" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the form:

```
$('#target').submit(function() {
 alert('Handler for .submit() called.');
 return false;
});
```

Now when the form is submitted, the message is alerted. This happens prior to the actual submission, so we can cancel the submit action by calling `.preventDefault()` on the event object or by returning `false` from our handler. We can trigger the event manually when another element is clicked:

```
$('#other').click(function() {
 $('#target').submit();
});
```

After this code executes, clicks on Trigger the handler will also display the message. In addition, the default `submit` action on the form will be fired, so the form will be submitted.

The JavaScript `submit` event does not bubble in Internet Explorer. However, scripts that rely on event delegation with the `submit` event will work consistently across browsers as of jQuery 1.4, which has normalized the event's behavior.

Example

If you'd like to prevent forms from being submitted unless a flag variable is set, try:

```
$("form").submit(function() {
   if ($("input:first").val() == "correct") {
    $("span").text("Validated...").show();
    return true;
   }
   $("span").text("Not valid!").show().fadeOut(1000);
   return false;
  });
```

Example
If you'd like to prevent forms from being submitted unless a flag variable is set, try:

```
$("form").submit( function () {
 return this.some_flag_variable;
} );
```

Example
To trigger the submit event on the first form on the page, try:

```
$("form:first").submit();
```

select(handler(eventObject))
Bind an event handler to the "select" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('select', handler)` in the first two variations, and `.trigger('select')` in the third.

The `select` event is sent to an element when the user makes a text selection inside it. This event is limited to `<input type="text">` fields and `<textarea>` boxes.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Hello there" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the text input:

```
$('#target').select(function() {
 alert('Handler for .select() called.');
});
```

Now when any portion of the text is selected, the alert is displayed. Merely setting the location of the insertion point will not trigger the event. To trigger the event manually, apply `.select()` without an argument:

```
$('#other').click(function() {
 $('#target').select();
});
```

After this code executes, clicks on the Trigger button will also alert the message:

Handler for .select() called.

In addition, the default `select` action on the field will be fired, so the entire text field will be selected.

The method for retrieving the current selected text differs from one browser to another. A number of jQuery plug-ins offer cross-platform solutions.

Example
To do something when text in input boxes is selected:

```
$(":input").select( function () {
   $("div").text("Something was selected").show().fadeOut(1000);
  });
```

Example
To trigger the select event on all input elements, try:

```
$("input").select();
```

change(handler(eventObject))
Bind an event handler to the "change" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('change', handler)` in the first two variations, and `.trigger('change')` in the third.

The `change` event is sent to an element when its value changes. This event is limited to `<input>` elements, `<textarea>` boxes and `<select>` elements. For select boxes, checkboxes, and radio buttons, the event is fired immediately when the user makes a selection with the mouse, but for the other element types the event is deferred until the element loses focus.

For example, consider the HTML:

```
<form>
 <input class="target" type="text" value="Field 1" />
 <select class="target">
  <option value="option1" selected="selected">Option 1</option>
  <option value="option2">Option 2</option>
 </select>
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the text input and the select box:

```
$('.target').change(function() {
 alert('Handler for .change() called.');
});
```

Now when the second option is selected from the dropdown, the alert is displayed. It is also displayed if you change the text in the field and then click away. If the field loses focus without the contents having changed, though, the event is not triggered. To trigger the event manually, apply `.change()` without arguments:

```
$('#other').click(function() {
 $('.target').change();
});
```

After this code executes, clicks on Trigger the handler will also alert the message. The message will display twice, because the handler has been bound to the `change` event on both of the form elements.

As of jQuery 1.4, the `change` event bubbles in Internet Explorer, behaving consistently with the event in other modern browsers.

Example
Attaches a change event to the select that gets the text for each selected option and writes them in the div. It then triggers the event for the initial text draw.

```
$("select").change(function () {
     var str = "";
```

```
    $("select option:selected").each(function () {
       str += $(this).text() + " ";
      });
    $("div").text(str);
   })
   .change();
```

Example
To add a validity test to all text input elements:

```
$("input[type='text']").change( function() {
 // check input ($(this).val()) for validity here
});
```

blur(handler(eventObject))
Bind an event handler to the "blur" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('blur', handler)` in the first two variations, and `.trigger('blur')` in the third.

The `blur` event is sent to an element when it loses focus. Originally, this event was only applicable to form elements, such as `<input>`. In recent browsers, the domain of the event has been extended to include all element types. An element can lose focus via keyboard commands, such as the Tab key, or by mouse clicks elsewhere on the page.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Field 1" />
 <input type="text" value="Field 2" />
</form>
<div id="other">
 Trigger the handler
</div>
The event handler can be bound to the first input field:
$('#target').blur(function() {
 alert('Handler for .blur() called.');
});
```

Now if the first field has the focus, clicking elsewhere or tabbing away from it displays the alert:

Handler for .blur() called.

To trigger the event programmatically, apply `.blur()` without an argument:

```
$('#other').click(function() {
 $('#target').blur();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `blur` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `blur` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `blur` to the `focusout` event in its event delegation methods, .live() and .delegate().

Example
To trigger the blur event on all paragraphs:

```
$("p").blur();
```

focus(handler(eventObject))

Bind an event handler to the "focus" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

• This method is a shortcut for `.bind('focus', handler)` in the first and second variations, and `.trigger('focus')` in the third.

• The `focus` event is sent to an element when it gains focus. This event is implicitly applicable to a limited set of elements, such as form elements (`<input>`, `<select>`, etc.) and links (`<a href>`). In recent browser versions, the event can be extended to include all element types by explicitly setting the element's `tabindex` property. An element can gain focus via keyboard commands, such as the Tab key, or by mouse clicks on the element.

• Elements with focus are usually highlighted in some way by the browser, for example with a dotted line surrounding the element. The focus is used to determine which element is the first to receive keyboard-related events.

Attempting to set focus to a hidden element causes an error in Internet Explorer. Take care to only use `.focus()` on elements that are visible. To run an element's focus event handlers without setting focus to the element, use `.triggerHandler("focus")` instead of `.focus()`.

For example, consider the HTML:

```
<form>
 <input id="target" type="text" value="Field 1" />
 <input type="text" value="Field 2" />
</form>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to the first input field:

```
$('#target').focus(function() {
 alert('Handler for .focus() called.');
});
```

Now clicking on the first field, or tabbing to it from another field, displays the alert:

Handler for .focus() called.

We can trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#target').focus();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `focus` event does not bubble in Internet Explorer. Therefore, scripts that rely on event delegation with the `focus` event will not work consistently across browsers. As of version 1.4.2, however, jQuery works around this limitation by mapping `focus` to the `focusin` event in its event delegation methods, .live() and .delegate().

Example

Fire focus.

```
$("input").focus(function () {
     $(this).next("span").css('display','inline').fadeOut(1000);
  });
```

Example

To stop people from writing in text input boxes, try:

```
$("input[type=text]").focus(function(){
 $(this).blur();
});
```

Example
To focus on a login input box with id 'login' on page startup, try:

```
$(document).ready(function(){
 $("#login").focus();
});
```

mousemove(handler(eventObject))
Bind an event handler to the "mousemove" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mousemove', handler)` in the first two variations, and `.trigger('mousemove')` in the third.

The `mousemove` event is sent to an element when the mouse pointer moves inside the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Move here
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to the target:

```
$("#target").mousemove(function(event) {
 var msg = "Handler for .mousemove() called at ";
 msg += event.pageX + ", " + event.pageY;
 $("#log").append("<div>" + msg + "</div>");
});
```

Now when the mouse pointer moves within the target button, the messages are appended to :

Handler for .mousemove() called at (399, 48)Handler for .mousemove() called at (398, 46)Handler for .mousemove() called at (397, 44)Handler for .mousemove() called at (396, 42)

To trigger the event manually, apply `.mousemove()` without an argument:

```
$("#other").click(function() {
 $("#target").mousemove();
});
```

After this code executes, clicks on the Trigger button will also append the message:

Handler for .mousemove() called at (undefined, undefined)

When tracking mouse movement, you usually need to know the actual position of the mouse pointer. The event object that is passed to the handler contains some information about the mouse coordinates. Properties such as `.clientX`, `.offsetX`, and `.pageX` are available, but support for them differs between browsers. Fortunately, jQuery normalizes the `.pageX` and `.pageY` properties so that they can be used in all browsers. These properties provide the X and Y coordinates of the mouse pointer relative to the top-left corner of the document, as illustrated in the example output above.

Keep in mind that the `mousemove` event is triggered whenever the mouse pointer moves, even for a pixel. This means that hundreds of events can be generated over a very small amount of time. If the handler has to do any significant processing, or if multiple handlers for the event exist, this can be a serious performance drain on the browser. It is important, therefore, to optimize `mousemove` handlers as much as possible, and to unbind them as soon as they are no longer needed.

A common pattern is to bind the `mousemove` handler from within a `mousedown` hander, and to unbind it from a corresponding `mouseup` handler. If

implementing this sequence of events, remember that the mouseup event might be sent to a different HTML element than the mousemove event was. To account for this, the mouseup handler should typically be bound to an element high up in the DOM tree, such as <body>.

Example
Show the mouse coordinates when the mouse is moved over the yellow div. Coordinates are relative to the window, which in this case is the iframe.

```
$("div").mousemove(function(e){
   var pageCoords = "( " + e.pageX + ", " + e.pageY + " )";
   var clientCoords = "( " + e.clientX + ", " + e.clientY + " )";
   $("span:first").text("( e.pageX, e.pageY ) : " + pageCoords);
   $("span:last").text("( e.clientX, e.clientY ) : " + clientCoords);
  });
```

mouseleave(handler(eventObject))
Bind an event handler to be fired when the mouse leaves an element, or trigger that handler on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for .bind('mouseleave', handler) in the first two variations, and .trigger('mouseleave') in the third.

The mouseleave JavaScript event is proprietary to Internet Explorer. Because of the event's general utility, jQuery simulates this event so that it can be used regardless of browser. This event is sent to an element when the mouse pointer leaves the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseleave(function() {
 $('#log').append('<div>Handler for .mouseleave() called.</div>');
});
```

Now when the mouse pointer moves out of the Outer , the message is appended to . You can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseleave();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

The mouseleave event differs from mouseout in the way it handles event bubbling. If mouseout were used in this example, then when the mouse pointer moved out of the Inner element, the handler would be triggered. This is usually undesirable behavior. The mouseleave event, on the other hand, only triggers its handler when the mouse leaves the element it is bound to, not a descendant. So in this example, the handler is triggered when the mouse leaves the Outer element, but not the Inner element.

Example
Show number of times mouseout and mouseleave events are triggered.mouseout fires when the pointer moves out of child element as well, while

mouseleave fires only when the pointer moves out of the bound element.

```
var i = 0;
  $("div.overout").mouseover(function(){
   $("p:first",this).text("mouse over");
  }).mouseout(function(){
   $("p:first",this).text("mouse out");
   $("p:last",this).text(++i);
  });

  var n = 0;
  $("div.enterleave").mouseenter(function(){
   $("p:first",this).text("mouse enter");
  }).mouseleave(function(){
   $("p:first",this).text("mouse leave");
   $("p:last",this).text(++n);
  });
```

mouseenter(handler(eventObject))

Bind an event handler to be fired when the mouse enters an element, or trigger that handler on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseenter', handler)` in the first two variations, and `.trigger('mouseenter')` in the third.

The `mouseenter` JavaScript event is proprietary to Internet Explorer. Because of the event's general utility, jQuery simulates this event so that it can be used regardless of browser. This event is sent to an element when the mouse pointer enters the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseenter(function() {
 $('#log').append('<div>Handler for .mouseenter() called.</div>');
});
```

Now when the mouse pointer moves over the Outer , the message is appended to . You can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseenter();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

The `mouseenter` event differs from `mouseover` in the way it handles event bubbling. If `mouseover` were used in this example, then when the mouse pointer moved over the Inner element, the handler would be triggered. This is usually undesirable behavior. The `mouseenter` event, on the other

hand, only triggers its handler when the mouse enters the element it is bound to, not a descendant. So in this example, the handler is triggered when the mouse enters the Outer element, but not the Inner element.

Example

Show texts when mouseenter and mouseout event triggering.`mouseover` fires when the pointer moves into the child element as well, while `mouseenter` fires only when the pointer moves into the bound element.

```
var i = 0;
  $("div.overout").mouseover(function(){
   $("p:first",this).text("mouse over");
   $("p:last",this).text(++i);
  }).mouseout(function(){
   $("p:first",this).text("mouse out");
  });

  var n = 0;
  $("div.enterleave").mouseenter(function(){
   $("p:first",this).text("mouse enter");
   $("p:last",this).text(++n);
  }).mouseleave(function(){
   $("p:first",this).text("mouse leave");
  });
```

mouseout(handler(eventObject))

Bind an event handler to the "mouseout" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseout', handler)` in the first two variation, and `.trigger('mouseout')` in the third.

The `mouseout` event is sent to an element when the mouse pointer leaves the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseout(function() {
 $('#log').append('Handler for .mouseout() called.');
});
```

Now when the mouse pointer moves out of the Outer`<div>`, the message is appended to `<div id="log">`. To trigger the event manually, apply `.mouseout()` without an argument::

```
$('#other').click(function() {
 $('#outer').mouseout();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

This event type can cause many headaches due to event bubbling. For instance, when the mouse pointer moves out of the Inner element in this example, a `mouseout` event will be sent to that, then trickle up to Outer. This can trigger the bound `mouseout` handler at inopportune times. See the discussion for `.mouseleave()` for a useful alternative.

Example

Show the number of times mouseout and mouseleave events are triggered.`mouseout` fires when the pointer moves out of the child element as well, while `mouseleave` fires only when the pointer moves out of the bound element.

```
var i = 0;
  $("div.overout").mouseout(function(){
   $("p:first",this).text("mouse out");
   $("p:last",this).text(++i);
  }).mouseover(function(){
   $("p:first",this).text("mouse over");
  });

  var n = 0;
  $("div.enterleave").bind("mouseenter",function(){
   $("p:first",this).text("mouse enter");
  }).bind("mouseleave",function(){
   $("p:first",this).text("mouse leave");
   $("p:last",this).text(++n);
  });
```

mouseover(handler(eventObject))

Bind an event handler to the "mouseover" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseover', handler)` in the first two variations, and `.trigger('mouseover')` in the third.

The `mouseover` event is sent to an element when the mouse pointer enters the element. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="outer">
 Outer
 <div id="inner">
  Inner
 </div>
</div>
<div id="other">
 Trigger the handler
</div>
<div id="log"></div>
```

The event handler can be bound to any element:

```
$('#outer').mouseover(function() {
 $('#log').append('<div>Handler for .mouseover() called.</div>');
});
```

Now when the mouse pointer moves over the Outer`<div>`, the message is appended to `<div id="log">`. We can also trigger the event when another element is clicked:

```
$('#other').click(function() {
 $('#outer').mouseover();
});
```

After this code executes, clicks on Trigger the handler will also append the message.

This event type can cause many headaches due to event bubbling. For instance, when the mouse pointer moves over the Inner element in this example, a `mouseover` event will be sent to that, then trickle up to Outer. This can trigger our bound `mouseover` handler at inopportune times. See the discussion for `.mouseenter()` for a useful alternative.

Example

Show the number of times mouseover and mouseenter events are triggered.`mouseover` fires when the pointer moves into the child element as well, while `mouseenter` fires only when the pointer moves into the bound element.

```
var i = 0;
 $("div.overout").mouseover(function() {
  i += 1;
  $(this).find("span").text( "mouse over x " + i );
 }).mouseout(function(){
  $(this).find("span").text("mouse out ");
 });

 var n = 0;
 $("div.enterleave").mouseenter(function() {
  n += 1;
  $(this).find("span").text( "mouse enter x " + n );
 }).mouseleave(function() {
  $(this).find("span").text("mouse leave");
 });
```

dblclick(handler(eventObject))

Bind an event handler to the "dblclick" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('dblclick', handler)` in the first two variations, and `.trigger('dblclick')` in the third.The `dblclick` event is sent to an element when the element is double-clicked. Any HTML element can receive this event.For example, consider the HTML:

```
<div id="target">
 Double-click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$('#target').dblclick(function() {
 alert('Handler for .dblclick() called.');
});
```

Now double-clicking on this element displays the alert:

Handler for .dblclick() called.

To trigger the event manually, apply `.dblclick()` without an argument:

```
$('#other').click(function() {
 $('#target').dblclick();
});
```

After this code executes, (single) clicks on Trigger the handler will also alert the message.

The dblclick event is only triggered after this exact series of events:

- The mouse button is depressed while the pointer is inside the element.
- The mouse button is released while the pointer is inside the element.
- The mouse button is depressed again while the pointer is inside the element, within a time window that is system-dependent.
- The mouse button is released while the pointer is inside the element.

It is inadvisable to bind handlers to both the click and dblclick events for the same element. The sequence of events triggered varies from browser to browser, with some receiving two click events before the dblclick and others only one. Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

Example

To bind a "Hello World!" alert box the dblclick event on every paragraph on the page:

```
$("p").dblclick( function () { alert("Hello World!"); });
```

Example

Double click to toggle background color.

```
var divdbl = $("div:first");
  divdbl.dblclick(function () {
   divdbl.toggleClass('dbl');
  });
```

click(handler(eventObject))

Bind an event handler to the "click" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

In the first two variations, this method is a shortcut for .bind("click", handler), as well as for .on("click", handler) as of jQuery 1.7. In the third variation, when .click() is called without arguments, it is a shortcut for .trigger("click").

The click event is sent to an element when the mouse pointer is over the element, and the mouse button is pressed and released. Any HTML element can receive this event.

```
For example, consider the HTML:
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any :

```
$("#target").click(function() {
 alert("Handler for .click() called.");
});
```

Now if we click on this element, the alert is displayed:

Handler for .click() called.

We can also trigger the event when a different element is clicked:

```
$("#other").click(function() {
 $("#target").click();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `click` event is only triggered after this exact series of events:

- The mouse button is depressed while the pointer is inside the element.
- The mouse button is released while the pointer is inside the element.

This is usually the desired sequence before taking an action. If this is not required, the `mousedown` or `mouseup` event may be more suitable.

Example

Hide paragraphs on a page when they are clicked:

```
$("p").click(function () {
   $(this).slideUp();
  });
```

Example

Trigger the click event on all of the paragraphs on the page:

```
$("p").click();
```

mouseup(handler(eventObject))

Bind an event handler to the "mouseup" JavaScript event, or trigger that event on an element.

Arguments

handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mouseup', handler)` in the first variation, and `.trigger('mouseup')` in the second.

The `mouseup` event is sent to an element when the mouse pointer is over the element, and the mouse button is released. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$('#target').mouseup(function() {
 alert('Handler for .mouseup() called.');
});
```

Now if we click on this element, the alert is displayed:

Handler for .mouseup() called.

We can also trigger the event when a different element is clicked:

```
$('#other').click(function() {
 $('#target').mouseup();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

If the user clicks outside an element, drags onto it, and releases the button, this is still counted as a `mouseup` event. This sequence of actions is not treated as a button press in most user interfaces, so it is usually better to use the `click` event unless we know that the `mouseup` event is preferable

for a particular situation.

Example
Show texts when mouseup and mousedown event triggering.

```
$("p").mouseup(function(){
   $(this).append('<span style="color:#F00;">Mouse up.</span>');
  }).mousedown(function(){
   $(this).append('<span style="color:#00F;">Mouse down.</span>');
  });
```

mousedown(handler(eventObject))
Bind an event handler to the "mousedown" JavaScript event, or trigger that event on an element.

Arguments
handler(eventObject) - A function to execute each time the event is triggered.

This method is a shortcut for `.bind('mousedown', handler)` in the first variation, and `.trigger('mousedown')` in the second.

The `mousedown` event is sent to an element when the mouse pointer is over the element, and the mouse button is pressed. Any HTML element can receive this event.

For example, consider the HTML:

```
<div id="target">
 Click here
</div>
<div id="other">
 Trigger the handler
</div>
```

The event handler can be bound to any `<div>`:

```
$('#target').mousedown(function() {
 alert('Handler for .mousedown() called.');
});
```

Now if we click on this element, the alert is displayed:

Handler for .mousedown() called.

We can also trigger the event when a different element is clicked:

```
$('#other').click(function() {
 $('#target').mousedown();
});
```

After this code executes, clicks on Trigger the handler will also alert the message.

The `mousedown` event is sent when any mouse button is clicked. To act only on specific buttons, we can use the event object's `which` property. Not all browsers support this property (Internet Explorer uses button instead), but jQuery normalizes the property so that it is safe to use in any browser. The value of `which` will be 1 for the left button, 2 for the middle button, or 3 for the right button.

This event is primarily useful for ensuring that the primary button was used to begin a drag operation; if ignored, strange results can occur when the user attempts to use a context menu. While the middle and right buttons can be detected with these properties, this is not reliable. In Opera and Safari, for example, right mouse button clicks are not detectable by default.

If the user clicks on an element, drags away from it, and releases the button, this is still counted as a `mousedown` event. This sequence of actions is treated as a "canceling" of the button press in most user interfaces, so it is usually better to use the `click` event unless we know that the `mousedown` event is preferable for a particular situation.

Example
Show texts when mouseup and mousedown event triggering.

```
$("p").mouseup(function(){
   $(this).append('<span style="color:#F00;">Mouse up.</span>');
  }).mousedown(function(){
   $(this).append('<span style="color:#00F;">Mouse down.</span>');
  });
```

error(handler(eventObject))
Bind an event handler to the "error" JavaScript event.

Arguments
handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for `.bind('error', handler)`.

The `error` event is sent to elements, such as images, that are referenced by a document and loaded by the browser. It is called if the element was not loaded correctly.

For example, consider a page with a simple image element:

```
<img alt="Book" id="book" />
```

The event handler can be bound to the image:

```
$('#book')
 .error(function() {
  alert('Handler for .error() called.')
 })
 .attr("src", "missing.png");
```

If the image cannot be loaded (for example, because it is not present at the supplied URL), the alert is displayed:

Handler for .error() called.

The event handler *must* be attached before the browser fires the error event, which is why the example sets the src attribute after attaching the handler. Also, the error event may not be correctly fired when the page is served locally; `error` relies on HTTP status codes and will generally not be triggered if the URL uses the `file:` protocol.

Note: A jQuery error event handler should not be attached to the window object. The browser fires the window's error event when a script error occurs. However, the window error event receives different arguments and has different return value requirements than conventional event handlers. Use `window.onerror` instead.

Example
To hide the "broken image" icons for IE users, you can try:

```
$("img")
 .error(function(){
  $(this).hide();
 })
 .attr("src", "missing.png");
```

unload(handler(eventObject))
Bind an event handler to the "unload" JavaScript event.

Arguments
handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for `.bind('unload', handler)`.

The unload event is sent to the window element when the user navigates away from the page. This could mean one of many things. The user could have clicked on a link to leave the page, or typed in a new URL in the address bar. The forward and back buttons will trigger the event. Closing the browser window will cause the event to be triggered. Even a page reload will first create an unload event.

The exact handling of the unload event has varied from version to version of browsers. For example, some versions of Firefox trigger the event when a link is followed, but not when the window is closed. In practical usage, behavior should be tested on all supported browsers, and contrasted with the proprietary beforeunload event.

Any unload event handler should be bound to the window object:

```
$(window).unload(function() {
 alert('Handler for .unload() called.');
});
```

After this code executes, the alert will be displayed whenever the browser leaves the current page.It is not possible to cancel the unload event with .preventDefault(). This event is available so that scripts can perform cleanup when the user leaves the page.

Example
To display an alert when a page is unloaded:

```
$(window).unload( function () { alert("Bye now!"); } );
```

load(handler(eventObject))
Bind an event handler to the "load" JavaScript event.

Arguments
handler(eventObject) - A function to execute when the event is triggered.

This method is a shortcut for .bind('load', handler).

The load event is sent to an element when it and all sub-elements have been completely loaded. This event can be sent to any element associated with a URL: images, scripts, frames, iframes, and the window object.

For example, consider a page with a simple image:

```
<img src="book.png" alt="Book" id="book" />
```

The event handler can be bound to the image:

```
$('#book').load(function() {
 // Handler for .load() called.
});
```

As soon as the image has been loaded, the handler is called.

In general, it is not necessary to wait for all images to be fully loaded. If code can be executed earlier, it is usually best to place it in a handler sent to the .ready() method.

The Ajax module also has a method named .load(). Which one is fired depends on the set of arguments passed.

Caveats of the load event when used with images

A common challenge developers attempt to solve using the .load() shortcut is to execute a function when an image (or collection of images) have completely loaded. There are several known caveats with this that should be noted. These are:

- It doesn't work consistently nor reliably cross-browser
- It doesn't fire correctly in WebKit if the image src is set to the same src as before
- It doesn't correctly bubble up the DOM tree
- Can cease to fire for images that already live in the browser's cache

Note: The .live() and .delegate() methods cannot be used to detect the load event of an iframe. The load event does not correctly bubble up the parent document and the event.target isn't set by Firefox, IE9 or Chrome, which is required to do event delegation.

Example
Run a function when the page is fully loaded including graphics.

```
$(window).load(function () {
 // run code
});
```

Example
Add the class bigImg to all images with height greater then 100 upon each image load.

```
$('img.userIcon').load(function(){
 if($(this).height() > 100) {
  $(this).addClass('bigImg');
 }
});
```

die()
Remove all event handlers previously attached using `.live()` from the elements.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to calling `.unbind()` with no arguments, which is used to remove all handlers attached with `.bind()`.See the discussions of `.live()` and `.unbind()` for further details.

As of jQuery 1.7, use of `.die()` (and its complementary method, `.live()`) is not recommended. Instead, use .off() to remove event handlers bound with .on()

Note: In order for .die() to function correctly, the selector used with it must match exactly the selector initially used with .live().

die(eventType, [handler])
Remove an event handler previously attached using `.live()` from the elements.

Arguments
eventType - A string containing a JavaScript event type, such as `click` or `keydown`.
handler - The function that is no longer to be executed.

Any handler that has been attached with `.live()` can be removed with `.die()`. This method is analogous to `.unbind()`, which is used to remove handlers attached with `.bind()`.See the discussions of `.live()` and `.unbind()` for further details.

Note: In order for `.die()` to function correctly, the selector used with it must match exactly the selector initially used with `.live()`.

Example
Can bind and unbind events to the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("#theone").live("click", aClick)
       .text("Can Click!");
});
$("#unbind").click(function () {
 $("#theone").die("click", aClick)
       .text("Does nothing...");
});
```

Example
To unbind all live events from all paragraphs, write:

```
$("p").die()
```

Example

To unbind all live click events from all paragraphs, write:

```
$("p").die( "click" )
```

Example
To unbind just one previously bound handler, pass the function in as the second argument:

```
var foo = function () {
// code to handle some kind of event
};

$("p").live("click", foo); // ... now foo will be called when paragraphs are clicked ...

$("p").die("click", foo); // ... foo will no longer be called.
```

unbind([eventType], [handler(eventObject)])
Remove a previously-attached event handler from the elements.

Arguments
eventType - A string containing a JavaScript event type, such as `click` or `submit`.
handler(eventObject) - The function that is to be no longer executed.

Event handlers attached with `.bind()` can be removed with `.unbind()`. (As of jQuery 1.7, the .on() and .off() methods are preferred to attach and remove event handlers on elements.) In the simplest case, with no arguments, `.unbind()` removes all handlers attached to the elements:

```
$('#foo').unbind();
```

This version removes the handlers regardless of type. To be more precise, we can pass an event type:

```
$('#foo').unbind('click');
```

By specifying the `click` event type, only handlers for that event type will be unbound. This approach can still have negative ramifications if other scripts might be attaching behaviors to the same element, however. Robust and extensible applications typically demand the two-argument version for this reason:

```
var handler = function() {
 alert('The quick brown fox jumps over the lazy dog.');
};
$('#foo').bind('click', handler);
$('#foo').unbind('click', handler);
```

By naming the handler, we can be assured that no other functions are accidentally removed. Note that the following will *not* work:

```
$('#foo').bind('click', function() {
 alert('The quick brown fox jumps over the lazy dog.');
});

// will NOT work
$('#foo').unbind('click', function() {
 alert('The quick brown fox jumps over the lazy dog.');
});
```

Even though the two functions are identical in content, they are created separately and so JavaScript is free to keep them as distinct function objects. To unbind a particular handler, we need a reference to that function and not a different one that happens to do the same thing.

Note: Using a proxied function to unbind an event on an element will unbind all proxied functions on that element, as the same proxy function is used for all proxied events. To allow unbinding a specific event, use unique class names on the event (e.g. `click.proxy1`, `click.proxy2`) when attaching them.

Using Namespaces

Instead of maintaining references to handlers in order to unbind them, we can namespace the events and use this capability to narrow the scope of our unbinding actions. As shown in the discussion for the `.bind()` method, namespaces are defined by using a period (`.`) character when binding a handler:

```
$('#foo').bind('click.myEvents', handler);
```

When a handler is bound in this fashion, we can still unbind it the normal way:

```
$('#foo').unbind('click');
```

However, if we want to avoid affecting other handlers, we can be more specific:

```
$('#foo').unbind('click.myEvents');
```

We can also unbind all of the handlers in a namespace, regardless of event type:

```
$('#foo').unbind('.myEvents');
```

It is particularly useful to attach namespaces to event bindings when we are developing plug-ins or otherwise writing code that may interact with other event-handling code in the future.

Using the Event Object

The third form of the `.unbind()` method is used when we wish to unbind a handler from within itself. For example, suppose we wish to trigger an event handler only three times:

```
var timesClicked = 0;
$('#foo').bind('click', function(event) {
 alert('The quick brown fox jumps over the lazy dog.');
 timesClicked++;
 if (timesClicked >= 3) {
  $(this).unbind(event);
 }
});
```

The handler in this case must take a parameter, so that we can capture the event object and use it to unbind the handler after the third click. The event object contains the context necessary for `.unbind()` to know which handler to remove.This example is also an illustration of a closure. Since the handler refers to the `timesClicked` variable, which is defined outside the function, incrementing the variable has an effect even between invocations of the handler.

Example

Can bind and unbind events to the colored button.

```
function aClick() {
$("div").show().fadeOut("slow");
}
$("#bind").click(function () {
// could use .bind('click', aClick) instead but for variety...
$("#theone").click(aClick)
 .text("Can Click!");
});
$("#unbind").click(function () {
$("#theone").unbind('click', aClick)
 .text("Does nothing...");
});
```

Example

To unbind all events from all paragraphs, write:

```
$("p").unbind()
```

Example
To unbind all click events from all paragraphs, write:

```
$("p").unbind( "click" )
```

Example
To unbind just one previously bound handler, pass the function in as the second argument:

```
var foo = function () {
// code to handle some kind of event
};

$("p").bind("click", foo); // ... now foo will be called when paragraphs are clicked ...

$("p").unbind("click", foo); // ... foo will no longer be called.
```

bind(eventType, [eventData], handler(eventObject))
Attach a handler to an event for the elements.

Arguments
eventType - A string containing one or more DOM event types, such as "click" or "submit," or custom event names.
eventData - A map of data that will be passed to the event handler.
handler(eventObject) - A function to execute each time the event is triggered.

As of jQuery 1.7, the .on() method is the preferred method for attaching event handlers to a document. For earlier versions, the `.bind()` method is used for attaching an event handler directly to elements. Handlers are attached to the currently selected elements in the jQuery object, so those elements *must exist* at the point the call to `.bind()` occurs. For more flexible event binding, see the discussion of event delegation in .on() or .delegate().

Any string is legal for `eventType`; if the string is not the name of a native DOM event, then the handler is bound to a custom event. These events are never called by the browser, but may be triggered manually from other JavaScript code using `.trigger()` or `.triggerHandler()`.

If the `eventType` string contains a period (`.`) character, then the event is namespaced. The period character separates the event from its namespace. For example, in the call `.bind('click.name', handler)`, the string `click` is the event type, and the string `name` is the namespace. Namespacing allows us to unbind or trigger some events of a type without affecting others. See the discussion of `.unbind()` for more information.

There are shorthand methods for some standard browser events such as .click() that can be used to attach or trigger event handlers. For a complete list of shorthand methods, see the events category.

When an event reaches an element, all handlers bound to that event type for the element are fired. If there are multiple handlers registered, they will always execute in the order in which they were bound. After all handlers have executed, the event continues along the normal event propagation path.

A basic usage of `.bind()` is:

```
$('#foo').bind('click', function() {
 alert('User clicked on "foo."');
});
```

This code will cause the element with an ID of `foo` to respond to the `click` event. When a user clicks inside this element thereafter, the alert will be shown.

Multiple Events

Multiple event types can be bound at once by including each one separated by a space:

```
$('#foo').bind('mouseenter mouseleave', function() {
 $(this).toggleClass('entered');
});
```

The effect of this on (when it does not initially have the "entered" class) is to add the "entered" class when the mouse enters the and remove the class when the mouse leaves.

As of jQuery 1.4 we can bind multiple event handlers simultaneously by passing a map of event type/handler pairs:

```
$('#foo').bind({
 click: function() {
  // do something on click
 },
 mouseenter: function() {
  // do something on mouseenter
 }
});
```

Event Handlers

The handler parameter takes a callback function, as shown above. Within the handler, the keyword this refers to the DOM element to which the handler is bound. To make use of the element in jQuery, it can be passed to the normal $() function. For example:

```
$('#foo').bind('click', function() {
 alert($(this).text());
});
```

After this code is executed, when the user clicks inside the element with an ID of foo, its text contents will be shown as an alert.

As of jQuery 1.4.2 duplicate event handlers can be bound to an element instead of being discarded. This is useful when the event data feature is being used, or when other unique data resides in a closure around the event handler function.

In jQuery 1.4.3 you can now pass in false in place of an event handler. This will bind an event handler equivalent to: function(){ return false; }. This function can be removed at a later time by calling: .unbind(eventName, false).

The Event object

The handler callback function can also take parameters. When the function is called, the event object will be passed to the first parameter.

The event object is often unnecessary and the parameter omitted, as sufficient context is usually available when the handler is bound to know exactly what needs to be done when the handler is triggered. However, at times it becomes necessary to gather more information about the user's environment at the time the event was initiated. View the full Event Object.

Returning false from a handler is equivalent to calling both .preventDefault() and .stopPropagation() on the event object.

Using the event object in a handler looks like this:

```
$(document).ready(function() {
 $('#foo').bind('click', function(event) {
  alert('The mouse cursor is at ('
   + event.pageX + ', ' + event.pageY + ')');
 });
});
```

Note the parameter added to the anonymous function. This code will cause a click on the element with ID foo to report the page coordinates of the mouse cursor at the time of the click.

Passing Event Data

The optional eventData parameter is not commonly used. When provided, this argument allows us to pass additional information to the handler. One

handy use of this parameter is to work around issues caused by closures. For example, suppose we have two event handlers that both refer to the same external variable:

```
var message = 'Spoon!';
$('#foo').bind('click', function() {
 alert(message);
});
message = 'Not in the face!';
$('#bar').bind('click', function() {
 alert(message);
});
```

Because the handlers are closures that both have `message` in their environment, both will display the message Not in the face! when triggered. The variable's value has changed. To sidestep this, we can pass the message in using `eventData`:

```
var message = 'Spoon!';
$('#foo').bind('click', {msg: message}, function(event) {
 alert(event.data.msg);
});
message = 'Not in the face!';
$('#bar').bind('click', {msg: message}, function(event) {
 alert(event.data.msg);
});
```

This time the variable is not referred to directly within the handlers; instead, the variable is passed in *by value* through `eventData`, which fixes the value at the time the event is bound. The first handler will now display Spoon! while the second will alert Not in the face!

Note that objects are passed to functions *by reference*, which further complicates this scenario.

If `eventData` is present, it is the second argument to the `.bind()` method; if no additional data needs to be sent to the handler, then the callback is passed as the second and final argument.

See the `.trigger()` method reference for a way to pass data to a handler at the time the event happens rather than when the handler is bound.

As of jQuery 1.4 we can no longer attach data (and thus, events) to object, embed, or applet elements because critical errors occur when attaching data to Java applets.

Note: Although demonstrated in the next example, it is inadvisable to bind handlers to both the `click` and `dblclick` events for the same element. The sequence of events triggered varies from browser to browser, with some receiving two click events before the `dblclick` and others only one. Double-click sensitivity (maximum time between clicks that is detected as a double click) can vary by operating system and browser, and is often user-configurable.

Example

Handle click and double-click for the paragraph. Note: the coordinates are window relative, so in this case relative to the demo iframe.

```
$("p").bind("click", function(event){
var str = "( " + event.pageX + ", " + event.pageY + " )";
$("span").text("Click happened! " + str);
});
$("p").bind("dblclick", function(){
$("span").text("Double-click happened in " + this.nodeName);
});
$("p").bind("mouseenter mouseleave", function(event){
$(this).toggleClass("over");
});
```

Example

To display each paragraph's text in an alert box whenever it is clicked:

```
$("p").bind("click", function(){
alert( $(this).text() );
```

```
});
```

Example
You can pass some extra data before the event handler:

```
function handler(event) {
alert(event.data.foo);
}
$("p").bind("click", {foo: "bar"}, handler)
```

Example
Cancel a default action and prevent it from bubbling up by returning `false`:

```
$("form").bind("submit", function() { return false; })
```

Example
Cancel only the default action by using the .preventDefault() method.

```
$("form").bind("submit", function(event) {
event.preventDefault();
});
```

Example
Stop an event from bubbling without preventing the default action by using the .stopPropagation() method.

```
$("form").bind("submit", function(event) {
 event.stopPropagation();
});
```

Example
Bind custom events.

```
$("p").bind("myCustomEvent", function(e, myName, myValue){
$(this).text(myName + ", hi there!");
$("span").stop().css("opacity", 1)
.text("myName = " + myName)
.fadeIn(30).fadeOut(1000);
});
$("button").click(function () {
$("p").trigger("myCustomEvent", [ "John" ]);
});
```

Example
Bind multiple events simultaneously.

```
$("div.test").bind({
 click: function(){
  $(this).addClass("active");
 },
 mouseenter: function(){
  $(this).addClass("inside");
 },
 mouseleave: function(){
  $(this).removeClass("inside");
 }
});
```

fadeTo(duration, opacity, [callback])
Adjust the opacity of the matched elements.

Arguments
duration - A string or number determining how long the animation will run.

opacity - A number between 0 and 1 denoting the target opacity.
callback - A function to call once the animation is complete.

The `.fadeTo()` method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, the default duration of `400` milliseconds is used. Unlike the other effect methods, `.fadeTo()` requires that `duration` be explicitly specified.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
  Click here
 </div>
 <img id="book" src="book.png" alt="" width="100" height="123" />
 With the element initially shown, we can dim it slowly:
 $('#clickme').click(function() {
  $('#book').fadeTo('slow', 0.5, function() {
   // Animation complete.
  });
 });
```

With `duration` set to `0`, this method just changes the `opacity` CSS property, so `.fadeTo(0, opacity)` is the same as `.css('opacity', opacity)`.

Example
Animates first paragraph to fade to an opacity of 0.33 (33%, about one third visible), completing the animation within 600 milliseconds.

```
$("p:first").click(function () {
$(this).fadeTo("slow", 0.33);
});
```

Example
Fade div to a random opacity on each click, completing the animation within 200 milliseconds.

```
$("div").click(function () {
$(this).fadeTo("fast", Math.random());
});
```

Example
Find the right answer! The fade will take 250 milliseconds and change various styles when it completes.

```
var getPos = function (n) {
return (Math.floor(n) * 90) + "px";
};
$("p").each(function (n) {
var r = Math.floor(Math.random() * 3);
var tmp = $(this).text();
$(this).text($("p:eq(" + r + ")").text());
$("p:eq(" + r + ")").text(tmp);
$(this).css("left", getPos(n));
});
$("div").each(function (n) {
   $(this).css("left", getPos(n));
  })
.css("cursor", "pointer")
```

```
.click(function () {
   $(this).fadeTo(250, 0.25, function () {
      $(this).css("cursor", "")
          .prev().css({"font-weight": "bolder",
                "font-style": "italic"});
     });
  });
```

fadeOut([duration], [callback])

Hide the matched elements by fading them to transparent.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.fadeOut()` method animates the opacity of the matched elements. Once the opacity reaches 0, the `display` style property is set to `none`, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially shown, we can hide it slowly:

```
$('#clickme').click(function() {
 $('#book').fadeOut('slow', function() {
  // Animation complete.
 });
});
```

Note: To avoid unnecessary DOM manipulation, `.fadeOut()` will not hide an element that is already considered hidden. For information on which elements jQuery considers hidden, see :hidden Selector.

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all paragraphs to fade out, completing the animation within 600 milliseconds.

```
$("p").click(function () {
 $("p").fadeOut("slow");
 });
```

Example
Fades out spans in one section that you click on.

```
$("span").click(function () {
 $(this).fadeOut(1000, function () {
 $("div").text("'" + $(this).text() + "' has faded!");
 $(this).remove();
 });
 });
 $("span").hover(function () {
 $(this).addClass("hilite");
 }, function () {
 $(this).removeClass("hilite");
 });
```

Example
Fades out two divs, one with a "linear" easing and one with the default, "swing," easing.

```
$("#btn1").click(function() {
 function complete() {
  $("<div/>").text(this.id).appendTo("#log");
 }

 $("#box1").fadeOut(1600, "linear", complete);
 $("#box2").fadeOut(1600, complete);
});

$("#btn2").click(function() {
 $("div").show();
 $("#log").empty();
});
```

fadeIn([duration], [callback])
Display the matched elements by fading them to opaque.

Arguments
duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.fadeIn()` method animates the opacity of the matched elements.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
   Click here
  </div>
  <img id="book" src="book.png" alt="" width="100" height="123" />
  With the element initially hidden, we can show it slowly:
  $('#clickme').click(function() {
   $('#book').fadeIn('slow', function() {
    // Animation complete
   });
  });
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates hidden divs to fade in one by one, completing each animation within 600 milliseconds.

```
$(document.body).click(function () {
    $("div:hidden:first").fadeIn("slow");
   });
```

Example

Fades a red block in over the text. Once the animation is done, it quickly fades in more text on top.

```
$("a").click(function () {
     $("div").fadeIn(3000, function () {
      $("span").fadeIn(100);
     });
     return false;
    });
```

slideToggle([duration], [callback])

Display or hide the matched elements with a sliding motion.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideToggle()` method animates the height of the matched elements. This causes lower parts of the page to slide up or down, appearing to reveal or conceal the items. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`. When the height reaches 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

We will cause `.slideToggle()` to be called when another element is clicked:

```
$('#clickme').click(function() {
 $('#book').slideToggle('slow', function() {
  // Animation complete.
```

```
 });
});
```

With the element initially shown, we can hide it slowly with the first click:

A second click will show the element once again:

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all paragraphs to slide up or down, completing the animation within 600 milliseconds.

```
$("button").click(function () {
   $("p").slideToggle("slow");
  });
```

Example

Animates divs between dividers with a toggle that makes some appear and some disappear.

```
$("#aa").click(function () {
  $("div:not(.still)").slideToggle("slow", function () {
   var n = parseInt($("span").text(), 10);
   $("span").text(n + 1);
  });
 });
```

slideUp([duration], [callback])

Hide the matched elements with a sliding motion.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideUp()` method animates the height of the matched elements. This causes lower parts of the page to slide up, appearing to conceal the items. Once the height reaches 0 (or, if set, to whatever the CSS min-height property is), the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially shown, we can hide it slowly:

```
$('#clickme').click(function() {
 $('#book').slideUp('slow', function() {
  // Animation complete.
 });
});
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all divs to slide up, completing the animation within 400 milliseconds.

```
$(document.body).click(function () {
  if ($("div:first").is(":hidden")) {
   $("div").show("slow");
  } else {
   $("div").slideUp();
  }
 });
```

Example

Animates the parent paragraph to slide up, completing the animation within 200 milliseconds. Once the animation is done, it displays an alert.

```
$("button").click(function () {
  $(this).parent().slideUp("slow", function () {
   $("#msg").text($("button", this).text() + " has completed.");
  });
 });
```

slideDown([duration], [callback])

Display the matched elements with a sliding motion.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

The `.slideDown()` method animates the height of the matched elements. This causes lower parts of the page to slide down, making way for the revealed items.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively. If any other string is supplied, or if the `duration` parameter is omitted, the default duration of `400` milliseconds is used.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

With the element initially hidden, we can show it slowly:

```
$('#clickme').click(function() {
 $('#book').slideDown('slow', function() {
  // Animation complete.
 });
});
```

Easing

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Animates all divs to slide down and show themselves over 600 milliseconds.

```
$(document.body).click(function () {
if ($("div:first").is(":hidden")) {
$("div").slideDown("slow");
} else {
$("div").hide();
}
});
```

Example

Animates all inputs to slide down, completing the animation within 1000 milliseconds. Once the animation is done, the input look is changed especially if it is the middle input which gets the focus.

```
$("div").click(function () {
$(this).css({ borderStyle:"inset", cursor:"wait" });
$("input").slideDown(1000,function(){
$(this).css("border", "2px red inset")
.filter(".middle")
 .css("background", "yellow")
 .focus();
$("div").css("visibility", "hidden");
});
});
```

toggle([duration], [callback])

Display or hide the matched elements.

Arguments

duration - A string or number determining how long the animation will run.
callback - A function to call once the animation is complete.

Note: The event handling suite also has a method named .toggle(). Which one is fired depends on the set of arguments passed.

With no parameters, the `.toggle()` method simply toggles the visibility of elements:

```
$('.target').toggle();
```

The matched elements will be revealed or hidden immediately, with no animation, by changing the CSS `display` property. If the element is initially displayed, it will be hidden; if hidden, it will be shown. The `display` property is saved and restored as needed. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration is provided, `.toggle()` becomes an animation method. The `.toggle()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0 after a hiding animation, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
```

We will cause `.toggle()` to be called when another element is clicked:

```
$('#clickme').click(function() {
 $('#book').toggle('slow', function() {
  // Animation complete.
 });
});
```

With the element initially shown, we can hide it slowly with the first click:

A second click will show the element once again:

The second version of the method accepts a Boolean parameter. If this parameter is `true`, then the matched elements are shown; if `false`, the elements are hidden. In essence, the statement:

```
$('#foo').toggle(showOrHide);
```

is equivalent to:

```
if ( showOrHide == true ) {
 $('#foo').show();
} else if ( showOrHide == false ) {
 $('#foo').hide();
}
```

Example
Toggles all paragraphs.

```
$("button").click(function () {
$("p").toggle();
});
```

Example
Animates all paragraphs to be shown if they are hidden and hidden if they are visible, completing the animation within 600 milliseconds.

```
$("button").click(function () {
$("p").toggle("slow");
});
```

Example
Shows all paragraphs, then hides them all, back and forth.

```
var flip = 0;
$("button").click(function () {
$("p").toggle( flip++ % 2 == 0 );
});
```

hide()
Hide the matched elements.

With no parameters, the `.hide()` method is the simplest way to hide an element:

```
$('.target').hide();
```

The matched elements will be hidden immediately, with no animation. This is roughly equivalent to calling `.css('display', 'none')`, except that the value of the `display` property is saved in jQuery's data cache so that `display` can later be restored to its initial value. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

When a duration is provided, `.hide()` becomes an animation method. The `.hide()` method animates the width, height, and opacity of the matched elements simultaneously. When these properties reach 0, the `display` style property is set to `none` to ensure that the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Note that `.hide()` is fired immediately and will override the animation queue if no duration or a duration of 0 is specified.

As of jQuery **1.4.3**, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
With the element initially shown, we can hide it slowly:
$('#clickme').click(function() {
 $('#book').hide('slow', function() {
  alert('Animation complete.');
 });
});
```

Example
Hides all paragraphs then the link on click.

```
$("p").hide();
  $("a").click(function ( event ) {
   event.preventDefault();
   $(this).hide();
  });
```

Example
Animates all shown paragraphs to hide slowly, completing the animation within 600 milliseconds.

```
$("button").click(function () {
   $("p").hide("slow");
  });
```

Example
Animates all spans (words in this case) to hide fastly, completing each animation within 200 milliseconds. Once each animation is done, it starts the next one.

```
$("#hidr").click(function () {
   $("span:last-child").hide("fast", function () {
    // use callee so don't have to name the function
    $(this).prev().hide("fast", arguments.callee);
   });
  });
  $("#showr").click(function () {
   $("span").show(2000);
  });
```

Example
Hides the divs when clicked over 2 seconds, then removes the div element when its hidden. Try clicking on more than one box at a time.

```
for (var i = 0; i < 5; i++) {
   $("<div>").appendTo(document.body);
  }
  $("div").click(function () {
   $(this).hide(2000, function () {
    $(this).remove();
   });
  });
```

show()
Display the matched elements.

With no parameters, the `.show()` method is the simplest way to display an element:

```
$('.target').show();
```

The matched elements will be revealed immediately, with no animation. This is roughly equivalent to calling `.css('display', 'block')`, except that the `display` property is restored to whatever it was initially. If an element has a `display` value of `inline`, then is hidden and shown, it will once again be displayed `inline`.

Note: If using !important in your styles, such as`display: none !important`,it is necessary to override the style using `.css('display', 'block !important')` should you wish for `.show()` to function correctly.

When a duration is provided, `.show()` becomes an animation method. The `.show()` method animates the width, height, and opacity of the matched elements simultaneously.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

As of jQuery 1.4.3, an optional string naming an easing function may be used. Easing functions specify the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

We can animate any element, such as a simple image:

```
<div id="clickme">
 Click here
</div>
<img id="book" src="book.png" alt="" width="100" height="123" />
With the element initially hidden, we can show it slowly:
$('#clickme').click(function() {
 $('#book').show('slow', function() {
  // Animation complete.
 });
});
```

Example

Animates all hidden paragraphs to show slowly, completing the animation within 600 milliseconds.

```
$("button").click(function () {
  $("p").show("slow");
  });
```

Example

Show the first div, followed by each next adjacent sibling div in order, with a 200ms animation. Each animation starts when the previous sibling div's animation ends.

```
$("#showr").click(function () {
 $("div").first().show("fast", function showNext() {
  $(this).next("div").show("fast", showNext);
 });
});

$("#hidr").click(function () {
 $("div").hide(1000);
});
```

Example

Show all span and input elements with an animation. Change the text once the animation is done.

```
function doIt() {
 $("span,div").show("slow");
}
```

```
/* can pass in function name */
$("button").click(doIt);

$("form").submit(function () {
 if ($("input").val() == "yes") {
  $("p").show(4000, function () {
   $(this).text("Ok, DONE! (now showing)");
  });
 }
 $("span,div").hide("fast");
 /* to stop the submit */
 return false;
});
```

Version 1.4.4

fadeToggle([duration], [easing], [callback])

Display or hide the matched elements by animating their opacity.

Arguments

duration - A string or number determining how long the animation will run.
easing - A string indicating which easing function to use for the transition.
callback - A function to call once the animation is complete.

The `.fadeToggle()` method animates the opacity of the matched elements. When called on a visible element, the element's `display` style property is set to `none` once the opacity reaches 0, so the element no longer affects the layout of the page.

Durations are given in milliseconds; higher values indicate slower animations, not faster ones. The strings `'fast'` and `'slow'` can be supplied to indicate durations of `200` and `600` milliseconds, respectively.

Easing

The string representing an easing function specifies the speed at which the animation progresses at different points within the animation. The only easing implementations in the jQuery library are the default, called `swing`, and one that progresses at a constant pace, called `linear`. More easing functions are available with the use of plug-ins, most notably the jQuery UI suite.

Callback Function

If supplied, the callback is fired once the animation is complete. This can be useful for stringing different animations together in sequence. The callback is not sent any arguments, but `this` is set to the DOM element being animated. If multiple elements are animated, it is important to note that the callback is executed once per matched element, not once for the animation as a whole.

As of jQuery 1.6, the `.promise()` method can be used in conjunction with the `deferred.done()` method to execute a single callback for the animation as a whole when *all* matching elements have completed their animations (See the example for .promise()).

Example

Fades first paragraph in or out, completing the animation within 600 milliseconds and using a linear easing. Fades last paragraph in or out for 200 milliseconds, inserting a "finished" message upon completion.

```
$("button:first").click(function() {
 $("p:first").fadeToggle("slow", "linear");
});
$("button:last").click(function () {
 $("p:last").fadeToggle("fast", function () {
  $("#log").append("<div>finished</div>");
 });
});
```

Version 1.5

jQuery.ajaxPrefilter([dataTypes], handler(options, originalOptions, jqXHR))

Handle custom Ajax options or modify existing options before each request is sent and before they are processed by `$.ajax()`.

Arguments

dataTypes - An optional string containing one or more space-separated dataTypes
handler(options, originalOptions, jqXHR) - A handler to set default values for future Ajax requests.

A typical prefilter registration using `$.ajaxPrefilter()` looks like this:

```
$.ajaxPrefilter( function( options, originalOptions, jqXHR ) {
 // Modify options, control originalOptions, store jqXHR, etc
});
```

where:

- `options` are the request options
- `originalOptions` are the options as provided to the ajax method, unmodified and, thus, without defaults from `ajaxSettings`
- `jqXHR` is the jqXHR object of the request

Prefilters are a perfect fit when custom options need to be handled. Given the following code, for example, a call to `$.ajax()` would automatically abort a request to the same URL if the custom `abortOnRetry` option is set to `true`:

```
var currentRequests = {};

$.ajaxPrefilter(function( options, originalOptions, jqXHR ) {
 if ( options.abortOnRetry ) {
  if ( currentRequests[ options.url ] ) {
   currentRequests[ options.url ].abort();
  }
  currentRequests[ options.url ] = jqXHR;
 }
});
```

Prefilters can also be used to modify existing options. For example, the following proxies cross-domain requests through http://mydomain.net/proxy/:

```
$.ajaxPrefilter( function( options ) {
 if ( options.crossDomain ) {
  options.url = "http://mydomain.net/proxy/" + encodeURIComponent( options.url );
  options.crossDomain = false;
 }
});
```

If the optional `dataTypes` argument is supplied, the prefilter will be only be applied to requests with the indicated dataTypes. For example, the following only applies the given prefilter to JSON and script requests:

```
$.ajaxPrefilter( "json script", function( options, originalOptions, jqXHR ) {
 // Modify options, control originalOptions, store jqXHR, etc
});
```

The `$.ajaxPrefilter()` method can also redirect a request to another dataType by returning that dataType. For example, the following sets a request as "script" if the URL has some specific properties defined in a custom `isActuallyScript()` function:

```
$.ajaxPrefilter(function( options ) {
 if ( isActuallyScript( options.url ) ) {
  return "script";
 }
});
```

This would ensure not only that the request is considered "script" but also that all the prefilters specifically attached to the script dataType would be applied to it.

jQuery.hasData(element)

Determine whether an element has any jQuery data associated with it.

Arguments

element - A DOM element to be checked for data.

The `jQuery.hasData()` method provides a way to determine if an element currently has any values that were set using `jQuery.data()`. If no data is associated with an element (there is no data object at all or the data object is empty), the method returns `false`; otherwise it returns `true`.

The primary advantage of `jQuery.hasData(element)` is that it does not create and associate a data object with the element if none currently exists. In contrast, `jQuery.data(element)` always returns a data object to the caller, creating one if no data object previously existed.

Note that jQuery's event system uses the jQuery data API to store event handlers. Therefore, binding an event to an element using `.on()`, `.bind()`, `.live()`, `.delegate()`, or one of the shorthand event methods also associates a data object with that element.

Example

Set data on an element and see the results of hasData.

```
var $p = jQuery("p"), p = $p[0];
$p.append(jQuery.hasData(p)+" "); /* false */

$.data(p, "testing", 123);
$p.append(jQuery.hasData(p)+" "); /* true*/

$.removeData(p, "testing");
$p.append(jQuery.hasData(p)+" "); /* false */

$p.on('click', function() {});
$p.append(jQuery.hasData(p)+" "); /* true */

$p.off('click');
$p.append(jQuery.hasData(p)+" "); /* false */
```

deferred.promise([target])

Return a Deferred's Promise object.

Arguments

target - Object onto which the promise methods have to be attached

The `deferred.promise()` method allows an asynchronous function to prevent other code from interfering with the progress or status of its internal request. The Promise exposes only the Deferred methods needed to attach additional handlers or determine the state (`then`, `done`, `fail`, `always`, `pipe`, `progress`, and `state`), but not ones that change the state (`resolve`, `reject`, `notify`, `resolveWith`, `rejectWith`, and `notifyWith`).

If `target` is provided, `deferred.promise()` will attach the methods onto it and then return this object rather than create a new one. This can be useful to attach the Promise behavior to an object that already exists.

If you are creating a Deferred, keep a reference to the Deferred so that it can be resolved or rejected at some point. Return *only* the Promise object via `deferred.promise()` so other code can register callbacks or inspect the current state.

For more information, see the documentation for Deferred object.

Example

Create a Deferred and set two timer-based functions to either resolve or reject the Deferred after a random interval. Whichever one fires first "wins" and will call one of the callbacks. The second timeout has no effect since the Deferred is already complete (in a resolved or rejected state) from the first timeout action. Also set a timer-based progress notification function, and call a progress handler that adds "working..." to the document body.

```
function asyncEvent(){
  var dfd = new jQuery.Deferred();

  // Resolve after a random interval
  setTimeout(function(){
    dfd.resolve("hurray");
  }, Math.floor(400+Math.random()*2000));

  // Reject after a random interval
  setTimeout(function(){
    dfd.reject("sorry");
  }, Math.floor(400+Math.random()*2000));

  // Show a "working..." message every half-second
  setTimeout(function working(){
    if ( dfd.state() === "pending" ) {
      dfd.notify("working... ");
      setTimeout(working, 500);
    }
  }, 1);

  // Return the Promise so caller can't change the Deferred
  return dfd.promise();
}

// Attach a done, fail, and progress handler for the asyncEvent
$.when( asyncEvent() ).then(
  function(status){
    alert( status+', things are going well' );
  },
  function(status){
    alert( status+', you fail this time' );
  },
  function(status){
    $("body").append(status);
  }
);
```

Example

Use the target argument to promote an existing object to a Promise:

```
// Existing object
var obj = {
 hello: function( name ) {
  alert( "Hello " + name );
 }
},
// Create a Deferred
defer = $.Deferred();

// Set object as a promise
defer.promise( obj );

// Resolve the deferred
defer.resolve( "John" );

// Use the object as a Promise
obj.done(function( name ) {
```

```
 obj.hello( name ); // will alert "Hello John"
}).hello( "Karl" ); // will alert "Hello Karl"
```

jQuery.parseXML(data)

Parses a string into an XML document.

Arguments

data - a well-formed XML string to be parsed

`jQuery.parseXML` uses the native parsing function of the browser to create a valid XML Document. This document can then be passed to `jQuery` to create a typical jQuery object that can be traversed and manipulated.

Example

Create a jQuery object using an XML string and obtain the value of the title node.

```
var xml = "<rss version='2.0'><channel><title>RSS Title</title></channel></rss>",
  xmlDoc = $.parseXML( xml ),
  $xml = $( xmlDoc ),
  $title = $xml.find( "title" );

/* append "RSS Title" to #someElement */
$( "#someElement" ).append( $title.text() );

/* change the title to "XML Title" */
$title.text( "XML Title" );

/* append "XML Title" to #anotherElement */
$( "#anotherElement" ).append( $title.text() );
```

jQuery.when(deferreds)

Provides a way to execute callback functions based on one or more objects, usually Deferred objects that represent asynchronous events.

Arguments

deferreds - One or more Deferred objects, or plain JavaScript objects.

If a single Deferred is passed to `jQuery.when`, its Promise object (a subset of the Deferred methods) is returned by the method. Additional methods of the Promise object can be called to attach callbacks, such as deferred.then. When the Deferred is resolved or rejected, usually by the code that created the Deferred originally, the appropriate callbacks will be called. For example, the jqXHR object returned by `jQuery.ajax` is a Deferred and can be used this way:

```
$.when( $.ajax("test.aspx") ).then(function(ajaxArgs){
   alert(ajaxArgs[1]); /* ajaxArgs is [ "success", statusText, jqXHR ] */
});
```

If a single argument is passed to `jQuery.when` and it is not a Deferred, it will be treated as a resolved Deferred and any doneCallbacks attached will be executed immediately. The doneCallbacks are passed the original argument. In this case any failCallbacks you might set are never called since the Deferred is never rejected. For example:

```
$.when( { testing: 123 } ).done(
  function(x){ alert(x.testing); } /* alerts "123" */
);
```

In the case where multiple Deferred objects are passed to `jQuery.when`, the method returns the Promise from a new "master" Deferred object that tracks the aggregate state of all the Deferreds it has been passed. The method will resolve its master Deferred as soon as all the Deferreds resolve, or reject the master Deferred as soon as one of the Deferreds is rejected. If the master Deferred is resolved, it is passed the resolved values of all the Deferreds that were passed to `jQuery.when`. For example, when the Deferreds are `jQuery.ajax()` requests, the arguments will be the jqXHR objects for the requests, in the order they were given in the argument list.

In the multiple-Deferreds case where one of the Deferreds is rejected, `jQuery.when` immediately fires the failCallbacks for its master Deferred. Note that some of the Deferreds may still be unresolved at that point. If you need to perform additional processing for this case, such as canceling any unfinished ajax requests, you can keep references to the underlying jqXHR objects in a closure and inspect/cancel them in the failCallback.

Example
Execute a function after two ajax requests are successful. (See the jQuery.ajax() documentation for a complete description of success and error cases for an ajax request).

```
$.when($.ajax("/page1.php"), $.ajax("/page2.php")).done(function(a1, a2){
  /* a1 and a2 are arguments resolved for the
    page1 and page2 ajax requests, respectively */
  var jqXHR = a1[2]; /* arguments are [ "success", statusText, jqXHR ] */
  if ( /Whip It/.test(jqXHR.responseText) ) {
   alert("First page has 'Whip It' somewhere.");
  }
});
```

Example
Execute the function `myFunc` when both ajax requests are successful, or `myFailure` if either one has an error.

```
$.when($.ajax("/page1.php"), $.ajax("/page2.php"))
 .then(myFunc, myFailure);
```

deferred.resolveWith(context, [args])
Resolve a Deferred object and call any doneCallbacks with the given `context` and `args`.

Arguments
context - Context passed to the doneCallbacks as the `this` object.
args - An optional array of arguments that are passed to the doneCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through deferred.promise().

When the Deferred is resolved, any doneCallbacks added by deferred.then or deferred.done are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.resolve()`. Any doneCallbacks added after the Deferred enters the resolved state are executed immediately when they are added, using the arguments that were passed to the `.resolve()` call. For more information, see the documentation for Deferred object.

deferred.rejectWith(context, [args])
Reject a Deferred object and call any failCallbacks with the given `context` and `args`.

Arguments
context - Context passed to the failCallbacks as the `this` object.
args - An optional array of arguments that are passed to the failCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through deferred.promise().

When the Deferred is rejected, any failCallbacks added by deferred.then or deferred.fail are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `deferred.reject()` call. Any failCallbacks added after the Deferred enters the rejected state are executed immediately when they are added, using the arguments that were passed to the `.reject()` call. For more information, see the documentation for Deferred object.

deferred.fail(failCallbacks, [failCallbacks])
Add handlers to be called when the Deferred object is rejected.

Arguments
failCallbacks - A function, or array of functions, that are called when the Deferred is rejected.
failCallbacks - Optional additional functions, or arrays of functions, that are called when the Deferred is rejected.

The `deferred.fail()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is rejected, the failCallbacks are called. Callbacks are executed in the order they were added. Since `deferred.fail()` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `deferred.fail()` methods. The failCallbacks are executed using the arguments provided to the deferred.reject() or deferred.rejectWith() method call in the order they were added. For more

information, see the documentation for Deferred object.

Example
Since the jQuery.get method returns a jqXHR object, which is derived from a Deferred, you can attach a success and failure callback using the `deferred.done()` and `deferred.fail()` methods.

```
$.get("test.php")
 .done(function(){ alert("$.get succeeded"); })
 .fail(function(){ alert("$.get failed!"); });
```

deferred.done(doneCallbacks, [doneCallbacks])
Add handlers to be called when the Deferred object is resolved.

Arguments
doneCallbacks - A function, or array of functions, that are called when the Deferred is resolved.
doneCallbacks - Optional additional functions, or arrays of functions, that are called when the Deferred is resolved.

The `deferred.done()` method accepts one or more arguments, all of which can be either a single function or an array of functions. When the Deferred is resolved, the doneCallbacks are called. Callbacks are executed in the order they were added. Since `deferred.done()` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `.done()` methods. When the Deferred is resolved, doneCallbacks are executed using the arguments provided to the resolve or resolveWith method call in the order they were added. For more information, see the documentation for Deferred object.

Example
Since the jQuery.get method returns a jqXHR object, which is derived from a Deferred object, we can attach a success callback using the `.done()` method.

```
$.get("test.php").done(function() {
 alert("$.get succeeded");
});
```

Example
Resolve a Deferred object when the user clicks a button, triggering a number of callback functions:

```
// 3 functions to call when the Deferred object is resolved
function fn1() {
 $("p").append(" 1 ");
}
function fn2() {
 $("p").append(" 2 ");
}
function fn3(n) {
 $("p").append(n + " 3 " + n);
}

// create a deferred object
var dfd = $.Deferred();

// add handlers to be called when dfd is resolved
dfd
// .done() can take any number of functions or arrays of functions
.done( [fn1, fn2], fn3, [fn2, fn1] )
// we can chain done methods, too
.done(function(n) {
 $("p").append(n + " we're done.");
});

// resolve the Deferred object when the button is clicked
$("button").bind("click", function() {
 dfd.resolve("and");
});
```

deferred.then(doneCallbacks, failCallbacks)

Add handlers to be called when the Deferred object is resolved or rejected.

Arguments

doneCallbacks - A function, or array of functions, called when the Deferred is resolved.
failCallbacks - A function, or array of functions, called when the Deferred is rejected.

All three arguments (including progressCallbacks, as of jQuery 1.7) can be either a single function or an array of functions. The arguments can also be `null` if no callback of that type is desired. Alternatively, use `.done()`, `.fail()` or `.progress()` to set only one type of callback.

When the Deferred is resolved, the doneCallbacks are called. If the Deferred is instead rejected, the failCallbacks are called. As of jQuery 1.7, the `deferred.notify()` or `deferred.notifyWith()` methods can be called to invoke the progressCallbacks as many times as desired before the Deferred is resolved or rejected.

Callbacks are executed in the order they were added. Since `deferred.then` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `.then()` methods. For more information, see the documentation for Deferred object.

Example

Since the jQuery.get method returns a jqXHR object, which is derived from a Deferred object, we can attach handlers using the `.then` method.

```
$.get("test.php").then(
  function(){ alert("$.get succeeded"); },
  function(){ alert("$.get failed!"); }
);
```

deferred.reject(args)

Reject a Deferred object and call any failCallbacks with the given `args`.

Arguments

args - Optional arguments that are passed to the failCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state by returning a restricted Promise object through deferred.promise().

When the Deferred is rejected, any failCallbacks added by deferred.then or deferred.fail are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `deferred.reject()` call. Any failCallbacks added after the Deferred enters the rejected state are executed immediately when they are added, using the arguments that were passed to the `.reject()` call. For more information, see the documentation for Deferred object.

deferred.isRejected()

Determine whether a Deferred object has been rejected.

As of jQuery 1.7 this API has been deprecated; please use deferred.state() instead.

Returns `true` if the Deferred object is in the rejected state, meaning that either deferred.reject() or deferred.rejectWith() has been called for the object and the failCallbacks have been called (or are in the process of being called).

Note that a Deferred object can be in one of three states: pending, resolved, or rejected; use deferred.isResolved() to determine whether the Deferred object is in the resolved state. These methods are primarily useful for debugging, for example to determine whether a Deferred has already been resolved even though you are inside code that intended to reject it.

deferred.isResolved()

Determine whether a Deferred object has been resolved.

As of jQuery 1.7 this API has been deprecated; please use deferred.state() instead.

Returns `true` if the Deferred object is in the resolved state, meaning that either deferred.resolve() or deferred.resolveWith() has been called for the object and the doneCallbacks have been called (or are in the process of being called).

Note that a Deferred object can be in one of three states: pending, resolved, or rejected; use deferred.isRejected() to determine whether the Deferred object is in the rejected state. These methods are primarily useful for debugging, for example to determine whether a Deferred has already been resolved even though you are inside code that intended to reject it.

deferred.resolve(args)

Resolve a Deferred object and call any doneCallbacks with the given `args`.

Arguments

args - Optional arguments that are passed to the doneCallbacks.

When the Deferred is resolved, any doneCallbacks added by deferred.then or deferred.done are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.resolve()`. Any doneCallbacks added after the Deferred enters the resolved state are executed immediately when they are added, using the arguments that were passed to the `.resolve()` call. For more information, see the documentation for Deferred object.

jQuery.sub()

Creates a new copy of jQuery whose properties and methods can be modified without affecting the original jQuery object.

This method is deprecated as of jQuery 1.7 and will be moved to a plugin in jQuery 1.8.

There are two specific use cases for which jQuery.sub() was created. The first was for providing a painless way of overriding jQuery methods without completely destroying the original methods and another was for helping to do encapsulation and basic namespacing for jQuery plugins.

Note that jQuery.sub() doesn't attempt to do any sort of isolation - that's not its intention. All the methods on the sub'd version of jQuery will still point to the original jQuery (events bound and triggered will still be through the main jQuery, data will be bound to elements through the main jQuery, Ajax queries and events will run through the main jQuery, etc.).

Note that if you're looking to use this for plugin development you should first *strongly* consider using something like the jQuery UI widget factory which manages both state and plugin sub-methods. Some examples of using the jQuery UI widget factory to build a plugin.

The particular use cases of this method can be best described through some examples.

Example

Adding a method to a jQuery sub so that it isn't exposed externally:

```
(function(){
  var sub$ = jQuery.sub();

  sub$.fn.myCustomMethod = function(){
   return 'just for me';
  };

  sub$(document).ready(function() {
   sub$('body').myCustomMethod() // 'just for me'
  });
 })();

 typeof jQuery('body').myCustomMethod // undefined
```

Example

Override some jQuery methods to provide new functionality.

```
(function() {
 var myjQuery = jQuery.sub();

 myjQuery.fn.remove = function() {
  // New functionality: Trigger a remove event
  this.trigger("remove");

  // Be sure to call the original jQuery remove method
```

```
  return jQuery.fn.remove.apply( this, arguments );
 };

 myjQuery(function($) {
  $(".menu").click(function() {
   $(this).find(".submenu").remove();
  });

  // A new remove event is now triggered from this copy of jQuery
  $(document).bind("remove", function(e) {
   $(e.target).parent().hide();
  });
 });
})();

// Regular jQuery doesn't trigger a remove event when removing an element
// This functionality is only contained within the modified 'myjQuery'.
```

Example

Create a plugin that returns plugin-specific methods.

```
(function() {
 // Create a new copy of jQuery using sub()
 var plugin = jQuery.sub();

 // Extend that copy with the new plugin methods
 plugin.fn.extend({
  open: function() {
   return this.show();
  },
  close: function() {
   return this.hide();
  }
 });

 // Add our plugin to the original jQuery
 jQuery.fn.myplugin = function() {
  this.addClass("plugin");

  // Make sure our plugin returns our special plugin version of jQuery
  return plugin( this );
 };
})();

$(document).ready(function() {
 // Call the plugin, open method now exists
 $('#main').myplugin().open();

 // Note: Calling just $("#main").open() won't work as open doesn't exist!
});
```

jQuery.post(url, [data], [success(data, textStatus, jqXHR)], [dataType])

Load data from the server using a HTTP POST request.

Arguments

url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.
dataType - The type of data expected from the server. Default: Intelligent Guess (xml, json, script, text, html).

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 type: 'POST',
 url: url,
 data: data,
 success: success,
 dataType: dataType
});
```

The `success` callback function is passed the returned data, which will be an XML root element or a text string depending on the MIME type of the response. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function is also passed a "jqXHR" object (in **jQuery 1.4**, it was passed the `XMLHttpRequest` object).

Most implementations will specify a success handler:

```
$.post('ajax/test.html', function(data) {
 $('.result').html(data);
});
```

This example fetches the requested HTML snippet and inserts it on the page.

Pages fetched with `POST` are never cached, so the `cache` and `ifModified` options in `jQuery.ajaxSetup()` have no effect on these requests.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.post()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.post()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
  // and remember the jqxhr object for this request
  var jqxhr = $.post("example.php", function() {
   alert("success");
  })
  .success(function() { alert("second success"); })
  .error(function() { alert("error"); })
  .complete(function() { alert("complete"); });

  // perform other work here ...

  // Set another completion function for the request above
  jqxhr.complete(function(){ alert("second complete"); });
```

Example

Request the test.php page, but ignore the return results.

```
$.post("test.php");
```

Example

Request the test.php page and send some additional data along (while still ignoring the return results).

```
$.post("test.php", { name: "John", time: "2pm" } );
```

Example

pass arrays of data to the server (while still ignoring the return results).

```
$.post("test.php", { 'choices[]': ["Jon", "Susan"] });
```

Example
send form data using ajax requests

```
$.post("test.php", $("#testform").serialize());
```

Example
Alert out the results from requesting test.php (HTML or XML, depending on what was returned).

```
$.post("test.php", function(data) {
  alert("Data Loaded: " + data);
 });
```

Example
Alert out the results from requesting test.php with an additional payload of data (HTML or XML, depending on what was returned).

```
$.post("test.php", { name: "John", time: "2pm" },
  function(data) {
   alert("Data Loaded: " + data);
  });
```

Example
Gets the test.php page content, store it in a XMLHttpResponse object and applies the process() JavaScript function.

```
$.post("test.php", { name: "John", time: "2pm" },
 function(data) {
  process(data);
 },
 "xml"
);
```

Example
Posts to the test.php page and gets contents which has been returned in json format (<?php echo json_encode(array("name"=>"John","time"=>"2pm")); ?>).

```
$.post("test.php", { "func": "getNameAndTime" },
 function(data){
  console.log(data.name); // John
  console.log(data.time); // 2pm
 }, "json");
```

Example
Post a form using ajax and put results in a div

```
/* attach a submit handler to the form */
 $("#searchForm").submit(function(event) {

  /* stop form from submitting normally */
  event.preventDefault();

  /* get some values from elements on the page: */
  var $form = $( this ),
    term = $form.find( 'input[name="s"]' ).val(),
    url = $form.attr( 'action' );

  /* Send the data using post and put the results in a div */
  $.post( url, { s: term },
   function( data ) {
     var content = $( data ).find( '#content' );
     $( "#result" ).empty().append( content );
```

```
   }
  );
 });
```

jQuery.getScript(url, [success(script, textStatus, jqXHR)])

Load a JavaScript file from the server using a GET HTTP request, then execute it.

Arguments

url - A string containing the URL to which the request is sent.
success(script, textStatus, jqXHR) - A callback function that is executed if the request succeeds.

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
 dataType: "script",
 success: success
});
```

The script is executed in the global context, so it can refer to other variables and use jQuery functions. Included scripts can have some impact on the current page.

Success Callback

The callback is passed the returned JavaScript file. This is generally not useful as the script will already have run at this point.

```
$(".result").html("<p>Lorem ipsum dolor sit amet.</p>");
```

Scripts are included and run by referencing the file name:

```
$.getScript("ajax/test.js", function(data, textStatus, jqxhr) {
  console.log(data); //data returned
  console.log(textStatus); //success
  console.log(jqxhr.status); //200
  console.log('Load was performed.');
});
```

Handling Errors

As of jQuery 1.5, you may use .fail() to account for errors:

```
$.getScript("ajax/test.js")
.done(function(script, textStatus) {
 console.log( textStatus );
})
.fail(function(jqxhr, settings, exception) {
 $( "div.log" ).text( "Triggered ajaxError handler." );
});
```

Prior to jQuery 1.5, the global `.ajaxError()` callback event had to be used in order to handle `$.getScript()` errors:

```
$( "div.log" ).ajaxError(function(e, jqxhr, settings, exception) {
 if (settings.dataType=='script') {
  $(this).text( "Triggered ajaxError handler." );
 }
});
```

Caching Responses

Be default, `$.getScript()` sets the cache setting to `false`. This appends a timestamped query parameter to the request URL to ensure that the browser downloads the script each time it is requested. You can override this feature by setting the cache property globally using $.ajaxSetup():

```
$.ajaxSetup({
 cache: true
});
```

Alternatively, you could define a new method that uses the more flexible `$.ajax()` method.

Example
Define a $.cachedScript() method that allows fetching a cached script:

```
jQuery.cachedScript = function(url, options) {

 // allow user to set any option except for dataType, cache, and url
 options = $.extend(options || {}, {
  dataType: "script",
  cache: true,
  url: url
 });

 // Use $.ajax() since it is more flexible than $.getScript
 // Return the jqXHR object so we can chain callbacks
 return jQuery.ajax(options);
};

// Usage
$.cachedScript("ajax/test.js").done(function(script, textStatus) {
 console.log( textStatus );
});
```

Example
Load the official jQuery Color Animation plugin dynamically and bind some color animations to occur once the new functionality is loaded.

```
$.getScript("/scripts/jquery.color.js", function() {
 $("#go").click(function(){
  $(".block").animate( { backgroundColor: "pink" }, 1000)
   .delay(500)
   .animate( { backgroundColor: "blue" }, 1000);
 });
});
```

jQuery.getJSON(url, [data], [success(data, textStatus, jqXHR)])
Load JSON-encoded data from the server using a GET HTTP request.

Arguments
url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
 dataType: 'json',
 data: data,
 success: callback
```

```
});
```

Data that is sent to the server is appended to the URL as a query string. If the value of the `data` parameter is an object (map), it is converted to a string and url-encoded before it is appended to the URL.

Most implementations will specify a success handler:

```
$.getJSON('ajax/test.json', function(data) {
 var items = [];

 $.each(data, function(key, val) {
  items.push('<li id="' + key + '">' + val + '</li>');
 });

 $('<ul/>', {
  'class': 'my-new-list',
  html: items.join('')
 }).appendTo('body');
});
```

This example, of course, relies on the structure of the JSON file:

```
{
 "one": "Singular sensation",
 "two": "Beady little eyes",
 "three": "Little birds pitch by my doorstep"
}
```

Using this structure, the example loops through the requested data, builds an unordered list, and appends it to the body.

The `success` callback is passed the returned data, which is typically a JavaScript object or array as defined by the JSON structure and parsed using the `$.parseJSON()` method. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function receives a "jqXHR" object (in **jQuery 1.4**, it received the `XMLHttpRequest` object). However, since JSONP and cross-domain GET requests do not use XHR, in those cases the `jqXHR` and `textStatus` parameters passed to the success callback are undefined.

Important: As of jQuery 1.4, if the JSON file contains a syntax error, the request will usually fail silently. Avoid frequent hand-editing of JSON data for this reason. JSON is a data-interchange format with syntax rules that are stricter than those of JavaScript's object literal notation. For example, all strings represented in JSON, whether they are properties or values, must be enclosed in double-quotes. For details on the JSON format, see http://json.org/.

JSONP

If the URL includes the string "callback=?" (or similar, as defined by the server-side API), the request is treated as JSONP instead. See the discussion of the `jsonp` data type in `$.ajax()` for more details.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.getJSON()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.getJSON()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.getJSON("example.json", function() {
 alert("success");
})
.success(function() { alert("second success"); })
.error(function() { alert("error"); })
.complete(function() { alert("complete"); });

// perform other work here ...

// Set another completion function for the request above
jqxhr.complete(function(){ alert("second complete"); });
```

Example
Loads the four most recent cat pictures from the Flickr JSONP API.

```
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?",
 {
  tags: "cat",
  tagmode: "any",
  format: "json"
 },
 function(data) {
  $.each(data.items, function(i,item){
   $("<img/>").attr("src", item.media.m).appendTo("#images");
   if ( i == 3 ) return false;
  });
 });
```

Example
Load the JSON data from test.js and access a name from the returned JSON data.

```
$.getJSON("test.js", function(json) {
  alert("JSON Data: " + json.users[3].name);
 });
```

Example
Load the JSON data from test.js, passing along additional data, and access a name from the returned JSON data.

```
$.getJSON("test.js", { name: "John", time: "2pm" }, function(json) {
  alert("JSON Data: " + json.users[3].name);
  });
```

jQuery.get(url, [data], [success(data, textStatus, jqXHR)], [dataType])
Load data from the server using a HTTP GET request.

Arguments
url - A string containing the URL to which the request is sent.
data - A map or string that is sent to the server with the request.
success(data, textStatus, jqXHR) - A callback function that is executed if the request succeeds.
dataType - The type of data expected from the server. Default: Intelligent Guess (xml, json, script, or html).

This is a shorthand Ajax function, which is equivalent to:

```
$.ajax({
 url: url,
 data: data,
 success: success,
 dataType: dataType
});
```

The `success` callback function is passed the returned data, which will be an XML root element, text string, JavaScript file, or JSON object, depending on the MIME type of the response. It is also passed the text status of the response.

As of jQuery 1.5, the `success` callback function is also passed a "jqXHR" object (in **jQuery 1.4**, it was passed the `XMLHttpRequest` object). However, since JSONP and cross-domain GET requests do not use XHR, in those cases the `(j)XHR` and `textStatus` parameters passed to the success callback are undefined.

Most implementations will specify a success handler:

```
$.get('ajax/test.html', function(data) {
 $('.result').html(data);
 alert('Load was performed.');
});
```

This example fetches the requested HTML snippet and inserts it on the page.

The jqXHR Object

As of jQuery 1.5, all of jQuery's Ajax methods return a superset of the `XMLHTTPRequest` object. This jQuery XHR object, or "jqXHR," returned by `$.get()` implements the Promise interface, giving it all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, it provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback.

The Promise interface in jQuery 1.5 also allows jQuery's Ajax methods, including `$.get()`, to chain multiple `.success()`, `.complete()`, and `.error()` callbacks on a single request, and even to assign these callbacks after the request may have completed. If the request is already complete, the callback is fired immediately.

```
// Assign handlers immediately after making the request,
 // and remember the jqxhr object for this request
 var jqxhr = $.get("example.php", function() {
  alert("success");
 })
 .success(function() { alert("second success"); })
 .error(function() { alert("error"); })
 .complete(function() { alert("complete"); });

 // perform other work here ...

 // Set another completion function for the request above
 jqxhr.complete(function(){ alert("second complete"); });
```

Example

Request the test.php page, but ignore the return results.

```
$.get("test.php");
```

Example

Request the test.php page and send some additional data along (while still ignoring the return results).

```
$.get("test.php", { name: "John", time: "2pm" } );
```

Example

pass arrays of data to the server (while still ignoring the return results).

```
$.get("test.php", { 'choices[]': ["Jon", "Susan"]} );
```

Example

Alert out the results from requesting test.php (HTML or XML, depending on what was returned).

```
$.get("test.php", function(data){
```

```
alert("Data Loaded: " + data);
});
```

Example
Alert out the results from requesting test.cgi with an additional payload of data (HTML or XML, depending on what was returned).

```
$.get("test.cgi", { name: "John", time: "2pm" },
  function(data){
   alert("Data Loaded: " + data);
  });
```

Example
Gets the test.php page contents, which has been returned in json format (<?php echo json_encode(array("name"=>"John","time"=>"2pm")); ?>), and adds it to the page.

```
$.get("test.php",
  function(data){
   $('body').append( "Name: " + data.name ) // John
        .append( "Time: " + data.time ); // 2pm
  }, "json");
```

jQuery.ajax(url, [settings])
Perform an asynchronous HTTP (Ajax) request.

Arguments
url - A string containing the URL to which the request is sent.
settings - A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup(). See jQuery.ajax(settings) below for a complete list of all settings.

The `$.ajax()` function underlies all Ajax requests sent by jQuery. It is often unnecessary to directly call this function, as several higher-level alternatives like `$.get()` and `.load()` are available and are easier to use. If less common options are required, though, `$.ajax()` can be used more flexibly.

At its simplest, the `$.ajax()` function can be called with no arguments:

```
$.ajax();
```

Note: Default settings can be set globally by using the `$.ajaxSetup()` function.

This example, using no options, loads the contents of the current page, but does nothing with the result. To use the result, we can implement one of the callback functions.

The jqXHR Object

The jQuery XMLHttpRequest (jqXHR) object returned by `$.ajax()` **as of jQuery 1.5** is a superset of the browser's native XMLHttpRequest object. For example, it contains `responseText` and `responseXML` properties, as well as a `getResponseHeader()` method. When the transport mechanism is something other than XMLHttpRequest (for example, a script tag for a JSONP request) the `jqXHR` object simulates native XHR functionality where possible.

As of jQuery 1.5.1, the `jqXHR` object also contains the `overrideMimeType()` method (it was available in jQuery 1.4.x, as well, but was temporarily removed in jQuery 1.5). The `.overrideMimeType()` method may be used in the `beforeSend()` callback function, for example, to modify the response content-type header:

```
$.ajax({
 url: "http://fiddle.jshell.net/favicon.png",
 beforeSend: function ( xhr ) {
  xhr.overrideMimeType("text/plain; charset=x-user-defined");
 }
}).done(function ( data ) {
 if( console && console.log ) {
```

```
  console.log("Sample of data:", data.slice(0, 100));
 }
});
```

The jqXHR objects returned by `$.ajax()` as of jQuery 1.5 implement the Promise interface, giving them all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, jqXHR also provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the `$.ajax()` request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback. This allows you to assign multiple callbacks on a single request, and even to assign callbacks after the request may have completed. (If the request is already complete, the callback is fired immediately.)

Deprecation Notice: The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callbacks will be deprecated in jQuery 1.8. To prepare your code for their eventual removal, use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.ajax( "example.php" )
  .done(function() { alert("success"); })
  .fail(function() { alert("error"); })
  .always(function() { alert("complete"); });

// perform other work here ...

// Set another completion function for the request above
jqxhr.always(function() { alert("second complete"); });
```

For backward compatibility with `XMLHttpRequest`, a `jqXHR` object will expose the following properties and methods:

- `readyState`
- `status`
- `statusText`
- `responseXML` and/or `responseText` when the underlying request responded with xml and/or text, respectively
- `setRequestHeader(name, value)` which departs from the standard by replacing the old value with the new one rather than concatenating the new value to the old one
- `getAllResponseHeaders()`
- `getResponseHeader()`
- `abort()`

No `onreadystatechange` mechanism is provided, however, since `success`, `error`, `complete` and `statusCode` cover all conceivable requirements.

Callback Function Queues

The `beforeSend`, `error`, `dataFilter`, `success` and `complete` options all accept callback functions that are invoked at the appropriate times.

As of jQuery 1.5, the `error` (`fail`), `success` (`done`), and `complete` (`always`, as of jQuery 1.6) callback hooks are first-in, first-out managed queues. This means you can assign more than one callback for each hook. See Deferred object methods, which are implemented internally for these `$.ajax()` callback hooks.

The `this` reference within all callbacks is the object in the `context` option passed to `$.ajax` in the settings; if `context` is not specified, `this` is a reference to the Ajax settings themselves.

Some types of Ajax requests, such as JSONP and cross-domain GET requests, do not use XHR; in those cases the `XMLHttpRequest` and `textStatus` parameters passed to the callback are `undefined`.

Here are the callback hooks provided by `$.ajax()`:

- `beforeSend` callback is invoked; it receives the `jqXHR` object and the `settings` map as parameters.
- `error` callbacks are invoked, in the order they are registered, if the request fails. They receive the `jqXHR`, a string indicating the error type, and an exception object if applicable. Some built-in errors will provide a string as the exception object: "abort", "timeout", "No Transport".
- `dataFilter` callback is invoked immediately upon successful receipt of response data. It receives the returned data and the value of `dataType`, and must return the (possibly altered) data to pass on to `success`.
- `success` callbacks are then invoked, in the order they are registered, if the request succeeds. They receive the returned data, a string containing

the success code, and the jqXHR object.

• complete callbacks fire, in the order they are registered, when the request finishes, whether in failure or success. They receive the jqXHR object, as well as a string containing the success or error code.

For example, to make use of the returned HTML, we can implement a success handler:

```
$.ajax({
 url: 'ajax/test.html',
 success: function(data) {
  $('.result').html(data);
  alert('Load was performed.');
 }
});
```

Data Types

The $.ajax() function relies on the server to provide information about the retrieved data. If the server reports the return data as XML, the result can be traversed using normal XML methods or jQuery's selectors. If another type is detected, such as HTML in the example above, the data is treated as text.

Different data handling can be achieved by using the dataType option. Besides plain xml, the dataType can be html, json, jsonp, script, or text.

The text and xml types return the data with no processing. The data is simply passed on to the success handler, either through the responseText or responseXML property of the jqXHR object, respectively.

Note: We must ensure that the MIME type reported by the web server matches our choice of dataType. In particular, XML must be declared by the server as text/xml or application/xml for consistent results.

If html is specified, any embedded JavaScript inside the retrieved data is executed before the HTML is returned as a string. Similarly, script will execute the JavaScript that is pulled back from the server, then return nothing.

The json type parses the fetched data file as a JavaScript object and returns the constructed object as the result data. To do so, it uses jQuery.parseJSON() when the browser supports it; otherwise it uses a Function**constructor**. Malformed JSON data will throw a parse error (see json.org for more information). JSON data is convenient for communicating structured data in a way that is concise and easy for JavaScript to parse. If the fetched data file exists on a remote server, specify the jsonp type instead.

The jsonp type appends a query string parameter of callback=? to the URL. The server should prepend the JSON data with the callback name to form a valid JSONP response. We can specify a parameter name other than callback with the jsonp option to $.ajax().

Note: JSONP is an extension of the JSON format, requiring some server-side code to detect and handle the query string parameter. More information about it can be found in the original post detailing its use.

When data is retrieved from remote servers (which is only possible using the script or jsonp data types), the error callbacks and global events will never be fired.

Sending Data to the Server

By default, Ajax requests are sent using the GET HTTP method. If the POST method is required, the method can be specified by setting a value for the type option. This option affects how the contents of the data option are sent to the server. POST data will always be transmitted to the server using UTF-8 charset, per the W3C XMLHTTPRequest standard.

The data option can contain either a query string of the form key1=value1&key2=value2, or a map of the form {key1: 'value1', key2: 'value2'}. If the latter form is used, the data is converted into a query string using jQuery.param() before it is sent. This processing can be circumvented by setting processData to false. The processing might be undesirable if you wish to send an XML object to the server; in this case, change the contentType option from application/x-www-form-urlencoded to a more appropriate MIME type.

Advanced Options

The global option prevents handlers registered using .ajaxSend(), .ajaxError(), and similar methods from firing when this request would

trigger them. This can be useful to, for example, suppress a loading indicator that was implemented with `.ajaxSend()` if the requests are frequent and brief. With cross-domain script and JSONP requests, the global option is automatically set to `false`. See the descriptions of these methods below for more details. See the descriptions of these methods below for more details.

If the server performs HTTP authentication before providing a response, the user name and password pair can be sent via the `username` and `password` options.

Ajax requests are time-limited, so errors can be caught and handled to provide a better user experience. Request timeouts are usually either left at their default or set as a global default using `$.ajaxSetup()` rather than being overridden for specific requests with the `timeout` option.

By default, requests are always issued, but the browser may serve results out of its cache. To disallow use of the cached results, set `cache` to `false`. To cause the request to report failure if the asset has not been modified since the last request, set `ifModified` to `true`.

The `scriptCharset` allows the character set to be explicitly specified for requests that use a `<script>` tag (that is, a type of `script` or `jsonp`). This is useful if the script and host page have differing character sets.

The first letter in Ajax stands for "asynchronous," meaning that the operation occurs in parallel and the order of completion is not guaranteed. The `async` option to `$.ajax()` defaults to `true`, indicating that code execution can continue after the request is made. Setting this option to `false` (and thus making the call no longer asynchronous) is strongly discouraged, as it can cause the browser to become unresponsive.

The `$.ajax()` function returns the `XMLHttpRequest` object that it creates. Normally jQuery handles the creation of this object internally, but a custom function for manufacturing one can be specified using the `xhr` option. The returned object can generally be discarded, but does provide a lower-level interface for observing and manipulating the request. In particular, calling `.abort()` on the object will halt the request before it completes.

At present, due to a bug in Firefox where `.getAllResponseHeaders()` returns the empty string although `.getResponseHeader('Content-Type')` returns a non-empty string, automatically decoding JSON CORS responses in Firefox with jQuery is not supported.

A workaround to this is possible by overriding `jQuery.ajaxSettings.xhr` as follows:

```
var _super = jQuery.ajaxSettings.xhr;
jQuery.ajaxSettings.xhr = function () {
  var xhr = _super(),
    getAllResponseHeaders = xhr.getAllResponseHeaders;

  xhr.getAllResponseHeaders = function () {
    if ( getAllResponseHeaders() ) {
      return getAllResponseHeaders();
    }
    var allHeaders = "";
    $( ["Cache-Control", "Content-Language", "Content-Type",
        "Expires", "Last-Modified", "Pragma"] ).each(function (i, header_name) {

      if ( xhr.getResponseHeader( header_name ) ) {
        allHeaders += header_name + ": " + xhr.getResponseHeader( header_name ) + "n";
      }
      return allHeaders;
    });
  };
  return xhr;
};
```

Extending Ajax

As of jQuery 1.5, jQuery's Ajax implementation includes prefilters, converters, and transports that allow you to extend Ajax with a great deal of flexibility. For more information about these advanced features, see the Extending Ajax page.

Example

Save some data to the server and notify the user once it's complete.

```
$.ajax({
 type: "POST",
 url: "some.php",
 data: { name: "John", location: "Boston" }
}).done(function( msg ) {
 alert( "Data Saved: " + msg );
});
```

Example
Retrieve the latest version of an HTML page.

```
$.ajax({
 url: "test.html",
 cache: false
}).done(function( html ) {
 $("#results").append(html);
});
```

Example
Send an xml document as data to the server. By setting the processData option to `false`, the automatic conversion of data to strings is prevented.

```
var xmlDocument = [create xml document];
var xmlRequest = $.ajax({
 url: "page.php",
 processData: false,
 data: xmlDocument
});

xmlRequest.done(handleResponse);
```

Example
Send an id as data to the server, save some data to the server, and notify the user once it's complete. If the request fails, alert the user.

```
var menuId = $("ul.nav").first().attr("id");
var request = $.ajax({
 url: "script.php",
 type: "POST",
 data: {id : menuId},
 dataType: "html"
});

request.done(function(msg) {
 $("#log").html( msg );
});

request.fail(function(jqXHR, textStatus) {
 alert( "Request failed: " + textStatus );
});
```

Example
Load and execute a JavaScript file.

```
$.ajax({
 type: "GET",
 url: "test.js",
 dataType: "script"
});
```

clone([withDataAndEvents])
Create a deep copy of the set of matched elements.

Arguments

withDataAndEvents - A Boolean indicating whether event handlers should be copied along with the elements. As of jQuery 1.4, element data will be copied as well.

The `.clone()` method performs a *deep* copy of the set of matched elements, meaning that it copies the matched elements as well as all of their descendant elements and text nodes. When used in conjunction with one of the insertion methods, `.clone()` is a convenient way to duplicate elements on a page. Consider the following HTML:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">Goodbye</div>
</div>
```

As shown in the discussion for `.append()`, normally when an element is inserted somewhere in the DOM, it is moved from its old location. So, given the code:

```
$('.hello').appendTo('.goodbye');
```

The resulting DOM structure would be:

```
<div class="container">
 <div class="goodbye">
  Goodbye
  <div class="hello">Hello</div>
 </div>
</div>
```

To prevent this and instead create a copy of the element, you could write the following:

```
$('.hello').clone().appendTo('.goodbye');
```

This would produce:

```
<div class="container">
 <div class="hello">Hello</div>
 <div class="goodbye">
  Goodbye
  <div class="hello">Hello</div>
 </div>
</div>
```

Note: When using the `.clone()` method, you can modify the cloned elements or their contents before (re-)inserting them into the document.

Normally, any event handlers bound to the original element are *not* copied to the clone. The optional `withDataAndEvents` parameter allows us to change this behavior, and to instead make copies of all of the event handlers as well, bound to the new copy of the element. As of jQuery 1.4, all element data (attached by the `.data()` method) is also copied to the new copy.

However, objects and arrays within element data are not copied and will continue to be shared between the cloned element and the original element. To deep copy all data, copy each one manually:

```
var $elem = $('#elem').data( "arr": [ 1 ] ), // Original element with attached data
  $clone = $elem.clone( true )
  .data( "arr", $.extend( [], $elem.data("arr") ) ); // Deep copy to prevent data sharing
```

As of jQuery 1.5, `withDataAndEvents` can be optionally enhanced with `deepWithDataAndEvents` to copy the events and data for all children of the cloned element.

Note: Using `.clone()` has the side-effect of producing elements with duplicate `id` attributes, which are supposed to be unique. Where possible, it is recommended to avoid cloning elements with this attribute or using `class` attributes as identifiers instead.

Example

Clones all b elements (and selects the clones) and prepends them to all paragraphs.

```
$("b").clone().prependTo("p");
```

Example
When using `.clone()` to clone a collection of elements that are not attached to the DOM, their order when inserted into the DOM is not guaranteed. However, it may be possible to preserve sort order with a workaround, as demonstrated:

```
// sort order is not guaranteed here and may vary with browser
$('#copy').append($('#orig .elem')
     .clone()
     .children('a')
     .prepend('foo - ')
     .parent()
     .clone());

// correct way to approach where order is maintained
$('#copy-correct')
     .append($('#orig .elem')
     .clone()
     .children('a')
     .prepend('bar - ')
     .end());
```

Version 1.5.1

jQuery.ajax(url, [settings])
Perform an asynchronous HTTP (Ajax) request.

Arguments
url - A string containing the URL to which the request is sent.
settings - A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup(). See jQuery.ajax(settings) below for a complete list of all settings.

The `$.ajax()` function underlies all Ajax requests sent by jQuery. It is often unnecessary to directly call this function, as several higher-level alternatives like `$.get()` and `.load()` are available and are easier to use. If less common options are required, though, `$.ajax()` can be used more flexibly.

At its simplest, the `$.ajax()` function can be called with no arguments:

```
$.ajax();
```

Note: Default settings can be set globally by using the `$.ajaxSetup()` function.

This example, using no options, loads the contents of the current page, but does nothing with the result. To use the result, we can implement one of the callback functions.

The jqXHR Object

The jQuery XMLHttpRequest (jqXHR) object returned by `$.ajax()`**as of jQuery 1.5** is a superset of the browser's native XMLHttpRequest object. For example, it contains `responseText` and `responseXML` properties, as well as a `getResponseHeader()` method. When the transport mechanism is something other than XMLHttpRequest (for example, a script tag for a JSONP request) the `jqXHR` object simulates native XHR functionality where possible.

As of jQuery 1.5.1, the `jqXHR` object also contains the `overrideMimeType()` method (it was available in jQuery 1.4.x, as well, but was temporarily removed in jQuery 1.5). The `.overrideMimeType()` method may be used in the `beforeSend()` callback function, for example, to modify the response content-type header:

```
$.ajax({
 url: "http://fiddle.jshell.net/favicon.png",
 beforeSend: function ( xhr ) {
```

```
  xhr.overrideMimeType("text/plain; charset=x-user-defined");
 }
}).done(function ( data ) {
 if( console && console.log ) {
  console.log("Sample of data:", data.slice(0, 100));
 }
});
```

The jqXHR objects returned by `$.ajax()` as of jQuery 1.5 implement the Promise interface, giving them all the properties, methods, and behavior of a Promise (see Deferred object for more information). For convenience and consistency with the callback names used by `$.ajax()`, jqXHR also provides `.error()`, `.success()`, and `.complete()` methods. These methods take a function argument that is called when the `$.ajax()` request terminates, and the function receives the same arguments as the correspondingly-named `$.ajax()` callback. This allows you to assign multiple callbacks on a single request, and even to assign callbacks after the request may have completed. (If the request is already complete, the callback is fired immediately.)

Deprecation Notice: The `jqXHR.success()`, `jqXHR.error()`, and `jqXHR.complete()` callbacks will be deprecated in jQuery 1.8. To prepare your code for their eventual removal, use `jqXHR.done()`, `jqXHR.fail()`, and `jqXHR.always()` instead.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.ajax( "example.php" )
  .done(function() { alert("success"); })
  .fail(function() { alert("error"); })
  .always(function() { alert("complete"); });

// perform other work here ...

// Set another completion function for the request above
jqxhr.always(function() { alert("second complete"); });
```

For backward compatibility with `XMLHttpRequest`, a `jqXHR` object will expose the following properties and methods:

- `readyState`
- `status`
- `statusText`
- `responseXML` and/or `responseText` when the underlying request responded with xml and/or text, respectively
- `setRequestHeader(name, value)` which departs from the standard by replacing the old value with the new one rather than concatenating the new value to the old one
- `getAllResponseHeaders()`
- `getResponseHeader()`
- `abort()`

No `onreadystatechange` mechanism is provided, however, since `success`, `error`, `complete` and `statusCode` cover all conceivable requirements.

Callback Function Queues

The `beforeSend`, `error`, `dataFilter`, `success` and `complete` options all accept callback functions that are invoked at the appropriate times.

As of jQuery 1.5, the `error` (`fail`), `success` (`done`), and `complete` (`always`, as of jQuery 1.6) callback hooks are first-in, first-out managed queues. This means you can assign more than one callback for each hook. See Deferred object methods, which are implemented internally for these `$.ajax()` callback hooks.

The `this` reference within all callbacks is the object in the `context` option passed to `$.ajax` in the settings; if `context` is not specified, `this` is a reference to the Ajax settings themselves.

Some types of Ajax requests, such as JSONP and cross-domain GET requests, do not use XHR; in those cases the `XMLHttpRequest` and `textStatus` parameters passed to the callback are `undefined`.

Here are the callback hooks provided by `$.ajax()`:

- `beforeSend` callback is invoked; it receives the `jqXHR` object and the `settings` map as parameters.
- `error` callbacks are invoked, in the order they are registered, if the request fails. They receive the `jqXHR`, a string indicating the error type, and

an exception object if applicable. Some built-in errors will provide a string as the exception object: "abort", "timeout", "No Transport".

• `dataFilter` callback is invoked immediately upon successful receipt of response data. It receives the returned data and the value of `dataType`, and must return the (possibly altered) data to pass on to `success`.

• `success` callbacks are then invoked, in the order they are registered, if the request succeeds. They receive the returned data, a string containing the success code, and the `jqXHR` object.

• `complete` callbacks fire, in the order they are registered, when the request finishes, whether in failure or success. They receive the `jqXHR` object, as well as a string containing the success or error code.

For example, to make use of the returned HTML, we can implement a `success` handler:

```
$.ajax({
 url: 'ajax/test.html',
 success: function(data) {
  $('.result').html(data);
  alert('Load was performed.');
 }
});
```

Data Types

The `$.ajax()` function relies on the server to provide information about the retrieved data. If the server reports the return data as XML, the result can be traversed using normal XML methods or jQuery's selectors. If another type is detected, such as HTML in the example above, the data is treated as text.

Different data handling can be achieved by using the `dataType` option. Besides plain `xml`, the `dataType` can be `html`, `json`, `jsonp`, `script`, or `text`.

The `text` and `xml` types return the data with no processing. The data is simply passed on to the success handler, either through the `responseText` or `responseXML` property of the `jqXHR` object, respectively.

Note: We must ensure that the MIME type reported by the web server matches our choice of `dataType`. In particular, XML must be declared by the server as `text/xml` or `application/xml` for consistent results.

If `html` is specified, any embedded JavaScript inside the retrieved data is executed before the HTML is returned as a string. Similarly, `script` will execute the JavaScript that is pulled back from the server, then return nothing.

The `json` type parses the fetched data file as a JavaScript object and returns the constructed object as the result data. To do so, it uses `jQuery.parseJSON()` when the browser supports it; otherwise it uses a `Function`**constructor**. Malformed JSON data will throw a parse error (see json.org for more information). JSON data is convenient for communicating structured data in a way that is concise and easy for JavaScript to parse. If the fetched data file exists on a remote server, specify the `jsonp` type instead.

The `jsonp` type appends a query string parameter of `callback=?` to the URL. The server should prepend the JSON data with the callback name to form a valid JSONP response. We can specify a parameter name other than `callback` with the `jsonp` option to `$.ajax()`.

Note: JSONP is an extension of the JSON format, requiring some server-side code to detect and handle the query string parameter. More information about it can be found in the original post detailing its use.

When data is retrieved from remote servers (which is only possible using the `script` or `jsonp` data types), the `error` callbacks and global events will never be fired.

Sending Data to the Server

By default, Ajax requests are sent using the GET HTTP method. If the POST method is required, the method can be specified by setting a value for the `type` option. This option affects how the contents of the `data` option are sent to the server. POST data will always be transmitted to the server using UTF-8 charset, per the W3C XMLHTTPRequest standard.

The `data` option can contain either a query string of the form `key1=value1&key2=value2`, or a map of the form `{key1: 'value1', key2: 'value2'}`. If the latter form is used, the data is converted into a query string using `jQuery.param()` before it is sent. This processing can be circumvented by setting `processData` to `false`. The processing might be undesirable if you wish to send an XML object to the server; in this case, change the `contentType` option from `application/x-www-form-urlencoded` to a more appropriate MIME type.

Advanced Options

The `global` option prevents handlers registered using `.ajaxSend()`, `.ajaxError()`, and similar methods from firing when this request would trigger them. This can be useful to, for example, suppress a loading indicator that was implemented with `.ajaxSend()` if the requests are frequent and brief. With cross-domain script and JSONP requests, the global option is automatically set to `false`. See the descriptions of these methods below for more details. See the descriptions of these methods below for more details.

If the server performs HTTP authentication before providing a response, the user name and password pair can be sent via the `username` and `password` options.

Ajax requests are time-limited, so errors can be caught and handled to provide a better user experience. Request timeouts are usually either left at their default or set as a global default using `$.ajaxSetup()` rather than being overridden for specific requests with the `timeout` option.

By default, requests are always issued, but the browser may serve results out of its cache. To disallow use of the cached results, set `cache` to `false`. To cause the request to report failure if the asset has not been modified since the last request, set `ifModified` to `true`.

The `scriptCharset` allows the character set to be explicitly specified for requests that use a `<script>` tag (that is, a type of `script` or `jsonp`). This is useful if the script and host page have differing character sets.

The first letter in Ajax stands for "asynchronous," meaning that the operation occurs in parallel and the order of completion is not guaranteed. The `async` option to `$.ajax()` defaults to `true`, indicating that code execution can continue after the request is made. Setting this option to `false` (and thus making the call no longer asynchronous) is strongly discouraged, as it can cause the browser to become unresponsive.

The `$.ajax()` function returns the `XMLHttpRequest` object that it creates. Normally jQuery handles the creation of this object internally, but a custom function for manufacturing one can be specified using the `xhr` option. The returned object can generally be discarded, but does provide a lower-level interface for observing and manipulating the request. In particular, calling `.abort()` on the object will halt the request before it completes.

At present, due to a bug in Firefox where `.getAllResponseHeaders()` returns the empty string although `.getResponseHeader('Content-Type')` returns a non-empty string, automatically decoding JSON CORS responses in Firefox with jQuery is not supported.

A workaround to this is possible by overriding `jQuery.ajaxSettings.xhr` as follows:

```
var _super = jQuery.ajaxSettings.xhr;
jQuery.ajaxSettings.xhr = function () {
  var xhr = _super(),
    getAllResponseHeaders = xhr.getAllResponseHeaders;

  xhr.getAllResponseHeaders = function () {
    if ( getAllResponseHeaders() ) {
      return getAllResponseHeaders();
    }
    var allHeaders = "";
    $( ["Cache-Control", "Content-Language", "Content-Type",
        "Expires", "Last-Modified", "Pragma"] ).each(function (i, header_name) {

      if ( xhr.getResponseHeader( header_name ) ) {
        allHeaders += header_name + ": " + xhr.getResponseHeader( header_name ) + "n";
      }
      return allHeaders;
    });
  };
  return xhr;
};
```

Extending Ajax

As of jQuery 1.5, jQuery's Ajax implementation includes prefilters, converters, and transports that allow you to extend Ajax with a great deal of flexibility. For more information about these advanced features, see the Extending Ajax page.

Example
Save some data to the server and notify the user once it's complete.

```
$.ajax({
 type: "POST",
 url: "some.php",
 data: { name: "John", location: "Boston" }
}).done(function( msg ) {
 alert( "Data Saved: " + msg );
});
```

Example
Retrieve the latest version of an HTML page.

```
$.ajax({
 url: "test.html",
 cache: false
}).done(function( html ) {
 $("#results").append(html);
});
```

Example
Send an xml document as data to the server. By setting the processData option to `false`, the automatic conversion of data to strings is prevented.

```
var xmlDocument = [create xml document];
var xmlRequest = $.ajax({
 url: "page.php",
 processData: false,
 data: xmlDocument
});

xmlRequest.done(handleResponse);
```

Example
Send an id as data to the server, save some data to the server, and notify the user once it's complete. If the request fails, alert the user.

```
var menuId = $("ul.nav").first().attr("id");
var request = $.ajax({
 url: "script.php",
 type: "POST",
 data: {id : menuId},
 dataType: "html"
});

request.done(function(msg) {
 $("#log").html( msg );
});

request.fail(function(jqXHR, textStatus) {
 alert( "Request failed: " + textStatus );
});
```

Example
Load and execute a JavaScript file.

```
$.ajax({
 type: "GET",
 url: "test.js",
 dataType: "script"
});
```

jQuery.support

A collection of properties that represent the presence of different browser features or bugs. Primarily intended for jQuery's internal use; specific properties may be removed when they are no longer needed internally to improve page startup performance.

Rather than using `$.browser` to detect the current user agent and alter the page presentation based on which browser is running, it is a good practice to perform **feature detection**. This means that prior to executing code which relies on a browser feature, we test to ensure that the feature works properly. To make this process simpler, jQuery performs many such tests and makes the results available to us as properties of the `jQuery.support` object.

The values of all the support properties are determined using feature detection (and do not use any form of browser sniffing).

Following are a few resources that explain how feature detection works:

- http://peter.michaux.ca/articles/feature-detection-state-of-the-art-browser-scripting
- http://www.jibbering.com/faq/faq_notes/not_browser_detect.html
- http://yura.thinkweb2.com/cft/

While jQuery includes a number of properties, developers should feel free to add their own as their needs dictate. Many of the `jQuery.support` properties are rather low-level, so they are most useful for plugin and jQuery core development, rather than general day-to-day development. Since jQuery requires these tests internally, they must be performed on *every* page load; for that reason this list is kept short and limited to features needed by jQuery itself.

The tests included in `jQuery.support` are as follows:

- `ajax` is equal to true if a browser is able to create an `XMLHttpRequest` object.
- `boxModel` is equal to true if the page is rendering according to the W3C CSS Box Model (is currently false in IE 6 and 7 when they are in Quirks Mode). This property is null until document ready occurs.
- `changeBubbles` is equal to true if the change event bubbles up the DOM tree, as required by the W3C DOM event model. (It is currently false in IE, and jQuery simulates bubbling).
- `checkClone` is equal to true if a browser correctly clones the checked state of radio buttons or checkboxes in document fragments.
- `checkOn` is equal to true if the value of a checkbox defaults to "on" when no value is specified.
- `cors` is equal to true if a browser can create an `XMLHttpRequest` object and if that `XMLHttpRequest` object has a `withCredentials` property. To enable cross-domain requests in environments that do not support cors yet but do allow cross-domain XHR requests (windows gadget, etc), set `$.support.cors = true;`.CORS WD
- `cssFloat` is equal to true if the name of the property containing the CSS float value is .cssFloat, as defined in the CSS Spec. (It is currently false in IE, it uses styleFloat instead).
- `hrefNormalized` is equal to true if the `.getAttribute()` method retrieves the `href` attribute of elements unchanged, rather than normalizing it to a fully-qualified URL. (It is currently false in IE, the URLs are normalized). DOM l3 spec
- `htmlSerialize` is equal to true if the browser is able to serialize/insert `<link>` elements using the `.innerHTML` property of elements. (is currently false in IE). HTML5 WD
- `leadingWhitespace` is equal to true if the browser inserts content with .innerHTML exactly as provided-specifically, if leading whitespace characters are preserved. (It is currently false in IE 6-8). HTML5 WD
- `noCloneChecked` is equal to true if cloned DOM elements copy over the state of the `.checked` expando. (It is currently false in IE). (Added in jQuery 1.5.1)
- `noCloneEvent` is equal to true if cloned DOM elements are created without event handlers (that is, if the event handlers on the source element are not cloned). (It is currently false in IE). DOM l2 spec
- `opacity` is equal to true if a browser can properly interpret the opacity style property. (It is currently false in IE, it uses alpha filters instead). CSS3 spec
- `optDisabled` is equal to true if option elements within disabled select elements are not automatically marked as disabled. HTML5 WD
- `optSelected` is equal to true if an `<option>` element that is selected by default has a working `selected` property. HTML5 WD
- `scriptEval()` is equal to true if inline scripts are automatically evaluated and executed when inserted into the document using standard DOM manipulation methods such as `.appendChild()` and `.createTextNode()`. (It is currently false in IE, it uses `.text` to insert executable scripts). **Note: No longer supported; removed in jQuery 1.6. Prior to jQuery 1.5.1**, the `scriptEval()` method was the static `scriptEval` property. The change to a method allowed the test to be deferred until first use to prevent content security policy inline-script violations. HTML5 WD
- `style` is equal to true if inline styles for an element can be accessed through the DOM attribute called style, as required by the DOM Level 2 specification. In this case, `.getAttribute('style')` can retrieve this value; in Internet Explorer, `.cssText` is used for this purpose. DOM l2 Style spec
- `submitBubbles` is equal to true if the submit event bubbles up the DOM tree, as required by the W3C DOM event model. (It is currently false in IE, and jQuery simulates bubbling).
- `tbody` is equal to true if an empty `<table>` element can exist without a `<tbody>` element. According to the HTML specification, this sub-element is optional, so the property should be true in a fully-compliant browser. If false, we must account for the possibility of the browser injecting `<tbody>` tags implicitly. (It is currently false in IE, which automatically inserts `tbody` if it is not present in a string assigned to `innerHTML`). HTML5 spec

Example

Returns the box model for the iframe.

```
$("p").html("This frame uses the W3C box model: <span>" +
        jQuery.support.boxModel + "</span>");
```

Example
Returns false if the page is in QuirksMode in Internet Explorer

```
jQuery.support.boxModel
```

Version 1.6

focus
Selects element if it is currently focused.

As with other pseudo-class selectors (those that begin with a ":"), it is recommended to precede `:focus` with a tag name or some other selector; otherwise, the universal selector ("*") is implied. In other words, the bare `$(':focus')` is equivalent to `$('*:focus')`. If you are looking for the currently focused element, `$( document.activeElement )` will retrieve it without having to search the whole DOM tree.

Example
Adds the focused class to whatever element has focus

```
$( "#content" ).delegate( "*", "focus blur", function( event ) {
  var elem = $( this );
  setTimeout(function() {
    elem.toggleClass( "focused", elem.is( ":focus" ) );
  }, 0);
});
```

deferred.pipe([doneFilter], [failFilter])
Utility method to filter and/or chain Deferreds.

Arguments
doneFilter - An optional function that is called when the Deferred is resolved.
failFilter - An optional function that is called when the Deferred is rejected.

The `deferred.pipe()` method returns a new promise that filters the status and values of a deferred through a function. The `doneFilter` and `failFilter` functions filter the original deferred's resolved / rejected status and values. **As of jQuery 1.7**, the method also accepts a `progressFilter` function to filter any calls to the original deferred's `notify` or `notifyWith` methods. These filter functions can return a new value to be passed along to the piped promise's `done()` or `fail()` callbacks, or they can return another observable object (Deferred, Promise, etc) which will pass its resolved / rejected status and values to the piped promise's callbacks. If the filter function used is `null`, or not specified, the piped promise will be resolved or rejected with the same values as the original.

Example
Filter resolve value:

```
var defer = $.Deferred(),
  filtered = defer.pipe(function( value ) {
   return value * 2;
  });

defer.resolve( 5 );
filtered.done(function( value ) {
 alert( "Value is ( 2*5 = ) 10: " + value );
});
```

Example
Filter reject value:

```
var defer = $.Deferred(),
  filtered = defer.pipe( null, function( value ) {
```

```
   return value * 3;
  });

defer.reject( 6 );
filtered.fail(function( value ) {
 alert( "Value is ( 3*6 = ) 18: " + value );
});
```

Example
Chain tasks:

```
var request = $.ajax( url, { dataType: "json" } ),
  chained = request.pipe(function( data ) {
   return $.ajax( url2, { data: { user: data.userId } } );
  });

chained.done(function( data ) {
 // data retrieved from url2 as provided by the first request
});
```

deferred.always(alwaysCallbacks, [alwaysCallbacks])
Add handlers to be called when the Deferred object is either resolved or rejected.

Arguments
alwaysCallbacks - A function, or array of functions, that is called when the Deferred is resolved or rejected.
alwaysCallbacks - Optional additional functions, or arrays of functions, that are called when the Deferred is resolved or rejected.

The argument can be either a single function or an array of functions. When the Deferred is resolved or rejected, the `alwaysCallbacks` are called. Since `deferred.always()` returns the Deferred object, other methods of the Deferred object can be chained to this one, including additional `.always()` methods. When the Deferred is resolved or rejected, callbacks are executed in the order they were added, using the arguments provided to the resolve, reject, resolveWith or rejectWith method calls. For more information, see the documentation for Deferred object.

Example
Since the jQuery.get() method returns a `jqXHR` object, which is derived from a Deferred object, we can attach a callback for both success and error using the `deferred.always()` method.

```
$.get("test.php").always( function() {
 alert("$.get completed with success or error callback arguments");
} );
```

promise([type], [target])
Return a Promise object to observe when all actions of a certain type bound to the collection, queued or not, have finished.

Arguments
type - The type of queue that needs to be observed.
target - Object onto which the promise methods have to be attached

The `.promise()` method returns a dynamically generated Promise that is resolved once all actions of a certain type bound to the collection, queued or not, have ended.

By default, `type` is `"fx"`, which means the returned Promise is resolved when all animations of the selected elements have completed.

Resolve context and sole argument is the collection onto which `.promise()` has been called.

If `target` is provided, `.promise()` will attach the methods onto it and then return this object rather than create a new one. This can be useful to attach the Promise behavior to an object that already exists.

Note: The returned Promise is linked to a Deferred object stored on the `.data()` for an element. Since the `.remove()` method removes the element's data as well as the element itself, it will prevent any of the element's unresolved Promises from resolving. If it is necessary to remove an element from the DOM before its Promise is resolved, use `.detach()` instead and follow with `.removeData()` after resolution.

Example
Using `.promise()` on a collection with no active animation returns a resolved Promise:

```
var div = $( "<div />" );

div.promise().done(function( arg1 ) {
 // will fire right away and alert "true"
 alert( this === div && arg1 === div );
});
```

Example
Resolve the returned Promise when all animations have ended (including those initiated in the animation callback or added later on):

```
$("button").bind( "click", function() {
 $("p").append( "Started...");

 $("div").each(function( i ) {
  $( this ).fadeIn().fadeOut( 1000 * (i+1) );
 });

 $( "div" ).promise().done(function() {
  $( "p" ).append( " Finished! " );
 });
});
```

Example
Resolve the returned Promise using a `$.when()` statement (the `.promise()` method makes it possible to do this with jQuery collections):

```
var effect = function() {
 return $("div").fadeIn(800).delay(1200).fadeOut();
};

$("button").bind( "click", function() {
 $("p").append( " Started... ");

 $.when( effect() ).done(function() {
  $("p").append(" Finished! ");
 });
});
```

removeProp(propertyName)
Remove a property for the set of matched elements.

Arguments
propertyName - The name of the property to set.

The `.removeProp()` method removes properties set by the `.prop()` method.

With some built-in properties of a DOM element or `window` object, browsers may generate an error if an attempt is made to remove the property. jQuery first assigns the value `undefined` to the property and ignores any error the browser generates. In general, it is only necessary to remove custom properties that have been set on an object, and not built-in (native) properties.

Note: Do not use this method to remove native properties such as checked, disabled, or selected. This will remove the property completely and, once removed, cannot be added again to element. Use `.prop()` to set these properties to `false` instead.

Example
Set a numeric property on a paragraph and then remove it.

```
var $para = $("p");
$para.prop("luggageCode", 1234);
$para.append("The secret luggage code is: ", String($para.prop("luggageCode")), ". ");
$para.removeProp("luggageCode");
```

```
$para.append("Now the secret luggage code is: ", String($para.prop("luggageCode")), ". ");
```

prop(propertyName)

Get the value of a property for the first element in the set of matched elements.

Arguments

propertyName - The name of the property to get.

The `.prop()` method gets the property value for only the *first* element in the matched set. It returns `undefined` for the value of a property that has not been set, or if the matched set has no elements. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

The difference between *attributes* and *properties* can be important in specific situations. **Before jQuery 1.6**, the `.attr()` method sometimes took property values into account when retrieving some attributes, which could cause inconsistent behavior. **As of jQuery 1.6**, the `.prop()` method provides a way to explicitly retrieve property values, while `.attr()` retrieves attributes.

For example, `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, and `defaultSelected` should be retrieved and set with the `.prop()` method. Prior to jQuery 1.6, these properties were retrievable with the `.attr()` method, but this was not within the scope of `attr`. These do not have corresponding attributes and are only properties.

Concerning boolean attributes, consider a DOM element defined by the HTML markup `<input type="checkbox" checked="checked" />`, and assume it is in a JavaScript variable named `elem`:

`elem.checked`	`true`	(Boolean) Will change with checkbox state	
`$(elem).prop("checked")`	`true`	(Boolean) Will change with checkbox state	
`elem.getAttribute("checked")`	`"checked"`	(String) Initial state of the checkbox; does not change	
`$(elem).attr("checked")`	*(1.6)*	`"checked"`	(String) Initial s
`$(elem).attr("checked")`	*(1.6.1+)*	`"checked"`	(String) Will ch
`$(elem).attr("checked")`	*(pre-1.6)*	`true`	(Boolean) Cha

According to the W3C forms specification, the `checked` attribute is a *boolean attribute*, which means the corresponding property is true if the attribute is present at all-even if, for example, the attribute has no value or an empty string value. The preferred cross-browser-compatible way to determine if a checkbox is checked is to check for a "truthy" value on the element's property using one of the following:

- `if ( elem.checked )`
- `if ( $(elem).prop("checked") )`
- `if ( $(elem).is(":checked") )`

If using jQuery 1.6, the code `if ( $(elem).attr("checked") )` will retrieve the actual content *attribute*, which does not change as the checkbox is checked and unchecked. It is meant only to store the default or initial value of the checked property. To maintain backwards compatability, the `.attr()` method in jQuery 1.6.1+ will retrieve and update the property for you so no code for boolean attributes is required to be changed to `.prop()`. Nevertheless, the preferred way to retrieve a checked value is with one of the options listed above. To see how this works in the latest jQuery, check/uncheck the checkbox in the example below.

Example

Display the checked property and attribute of a checkbox as it changes.

```
$("input").change(function() {
 var $input = $(this);
 $("p").html(".attr('checked'): <b>" + $input.attr('checked') + "</b><br>"
       + ".prop('checked'): <b>" + $input.prop('checked') + "</b><br>"
       + ".is(':checked'): <b>" + $input.is(':checked') ) + "</b>";
}).change();
```

prop(propertyName, value)

Set one or more properties for the set of matched elements.

Arguments

propertyName - The name of the property to set.
value - A value to set for the property.

The `.prop()` method is a convenient way to set the value of properties-especially when setting multiple properties, using values returned by a function, or setting values on multiple elements at once. It should be used when setting `selectedIndex`, `tagName`, `nodeName`, `nodeType`, `ownerDocument`, `defaultChecked`, or `defaultSelected`. Since jQuery 1.6, these properties can no longer be set with the `.attr()` method. They do not have corresponding attributes and are only properties.

Properties generally affect the dynamic state of a DOM element without changing the serialized HTML attribute. Examples include the `value` property of input elements, the `disabled` property of inputs and buttons, or the `checked` property of a checkbox. The `.prop()` method should be used to set disabled and checked instead of the `.attr()` method. The `.val()` method should be used for getting and setting value.

```
$("input").prop("disabled", false);
$("input").prop("checked", true);
$("input").val("someValue");
```

Important: the `.removeProp()` method should not be used to set these properties to false. Once a native property is removed, it cannot be added again. See `.removeProp()` for more information.

Computed property values

By using a function to set properties, you can compute the value based on other properties of the element. For example, to toggle all checkboxes based off their individual values:

```
$("input[type='checkbox']").prop("checked", function( i, val ) {
 return !val;
});
```

Note: If nothing is returned in the setter function (ie. `function(index, prop){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example
Disable all checkboxes on the page.

```
$("input[type='checkbox']").prop({
 disabled: true
});
```

jQuery.holdReady(hold)
Holds or releases the execution of jQuery's ready event.

Arguments
hold - Indicates whether the ready hold is being requested or released

The `$.holdReady()` method allows the caller to delay jQuery's ready event. This *advanced feature* would typically be used by dynamic script loaders that want to load additional JavaScript such as jQuery plugins before allowing the ready event to occur, even though the DOM may be ready. This method must be called early in the document, such as in the `<head>` immediately after the jQuery script tag. Calling this method after the ready event has already fired will have no effect.

To delay the ready event, first call `$.holdReady(true)`. When the ready event should be released to execute, call `$.holdReady(false)`. Note that multiple holds can be put on the ready event, one for each `$.holdReady(true)` call. The ready event will not actually fire until all holds have been released with a corresponding number of `$.holdReady(false)` calls *and* the normal document ready conditions are met. (See ready for more information.)

Example
Delay the ready event until a custom plugin has loaded.

```
$.holdReady(true);
$.getScript("myplugin.js", function() {
```

```
   $.holdReady(false);
});
```

undelegate()

Remove a handler from the event for all elements which match the current selector, based upon a specific set of root elements.

The `.undelegate()` method is a way of removing event handlers that have been bound using .delegate(). **As of jQuery 1.7**, the .on() and .off() methods are preferred for attaching and removing event handlers.

Example

Can bind and unbind events to the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("body").delegate("#theone", "click", aClick)
  .find("#theone").text("Can Click!");
});
$("#unbind").click(function () {
 $("body").undelegate("#theone", "click", aClick)
  .find("#theone").text("Does nothing...");
});
```

Example

To unbind all delegated events from all paragraphs, write:

```
$("p").undelegate()
```

Example

To unbind all delegated click events from all paragraphs, write:

```
$("p").undelegate( "click" )
```

Example

To undelegate just one previously bound handler, pass the function in as the third argument:

```
var foo = function () {
 // code to handle some kind of event
};

// ... now foo will be called when paragraphs are clicked ...
$("body").delegate("p", "click", foo);


// ... foo will no longer be called.
$("body").undelegate("p", "click", foo);
```

Example

To unbind all delegated events by their namespace:

```
var foo = function () {
 // code to handle some kind of event
};

// delegate events under the ".whatever" namespace
$("form").delegate(":button", "click.whatever", foo);

$("form").delegate("input[type='text']", "keypress.whatever", foo);

// unbind all events delegated under the ".whatever" namespace
```

```
$("form").undelegate(".whatever");
```

parentsUntil([selector], [filter])

Get the ancestors of each element in the current set of matched elements, up to but not including the element matched by the selector, DOM node, or jQuery object.

Arguments

selector - A string containing a selector expression to indicate where to stop matching ancestor elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.parentsUntil()` method traverses through the ancestors of these elements until it reaches an element matched by the selector passed in the method's argument. The resulting jQuery object contains all of the ancestors up to but not including the one matched by the `.parentsUntil()` selector.

If the selector is not matched or is not supplied, all ancestors will be selected; in these cases it selects the same elements as the `.parents()` method does when no selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be used for the first **.parentsUntil()** argument.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

Example

Find the ancestors of <li class="item-a"> up to <ul class="level-1"> and give them a red background color. Also, find ancestors of <li class="item-2"> that have a class of "yes" up to <ul class="level-1"> and give them a green border.

```
$("li.item-a").parentsUntil(".level-1")
 .css("background-color", "red");

$("li.item-2").parentsUntil( $("ul.level-1"), ".yes" )
 .css("border", "3px solid green");
```

prevUntil([selector], [filter])

Get all preceding siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object.

Arguments

selector - A string containing a selector expression to indicate where to stop matching preceding sibling elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.prevUntil()` method searches through the predecessors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all previous siblings up to but not including the one matched by the `.prevUntil()` selector; the elements are returned in order from the closest sibling to the farthest.

If the selector is not matched or is not supplied, all previous siblings will be selected; in these cases it selects the same elements as the `.prevAll()` method does when no filter selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be used for the first **.prevUntil()** argument.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

Example

Find the siblings that precede <dt id="term-2"> up to the preceding <dt> and give them a red background color. Also, find previous <dd> siblings of <dt id="term-3"> up to <dt id="term-1"> and give them a green text color.

```
$("#term-2").prevUntil("dt")
 .css("background-color", "red");

var term1 = document.getElementById('term-1');
```

```
$("#term-3").prevUntil(term1, "dd")
 .css("color", "green");
```

nextUntil([selector], [filter])

Get all following siblings of each element up to but not including the element matched by the selector, DOM node, or jQuery object passed.

Arguments

selector - A string containing a selector expression to indicate where to stop matching following sibling elements.
filter - A string containing a selector expression to match elements against.

Given a selector expression that represents a set of DOM elements, the `.nextUntil()` method searches through the successors of these elements in the DOM tree, stopping when it reaches an element matched by the method's argument. The new jQuery object that is returned contains all following siblings up to but not including the one matched by the `.nextUntil()` argument.

If the selector is not matched or is not supplied, all following siblings will be selected; in these cases it selects the same elements as the `.nextAll()` method does when no filter selector is provided.

As of jQuery 1.6, A DOM node or jQuery object, instead of a selector, may be passed to the `.nextUntil()` method.

The method optionally accepts a selector expression for its second argument. If this argument is supplied, the elements will be filtered by testing whether they match it.

Example

Find the siblings that follow <dt id="term-2"> up to the next <dt> and give them a red background color. Also, find <dd> siblings that follow <dt id="term-1"> up to <dt id="term-3"> and give them a green text color.

```
$("#term-2").nextUntil("dt")
 .css("background-color", "red");

var term3 = document.getElementById("term-3");
$("#term-1").nextUntil(term3, "dd")
 .css("color", "green");
```

find(selector)

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

Arguments

selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.find()` method allows us to search through the descendants of these elements in the DOM tree and construct a new jQuery object from the matching elements. The `.find()` and `.children()` methods are similar, except that the latter only travels a single level down the DOM tree.

The first signature for the `.find()`method accepts a selector expression of the same type that we can pass to the `$()` function. The elements will be filtered by testing whether they match this selector.

Consider a page with a basic nested list on it:

```
<ul class="level-1">
 <li class="item-i">I</li>
 <li class="item-ii">II
  <ul class="level-2">
   <li class="item-a">A</li>
   <li class="item-b">B
    <ul class="level-3">
     <li class="item-1">1</li>
     <li class="item-2">2</li>
     <li class="item-3">3</li>
    </ul>
   </li>
```

```
   <li class="item-c">C</li>
  </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

If we begin at item II, we can find list items within it:

```
$('li.item-ii').find('li').css('background-color', 'red');
```

The result of this call is a red background on items A, B, 1, 2, 3, and C. Even though item II matches the selector expression, it is not included in the results; only descendants are considered candidates for the match.

Unlike in the rest of the tree traversal methods, the selector expression is required in a call to `.find()`. If we need to retrieve all of the descendant elements, we can pass in the universal selector `'*'` to accomplish this.

Selector context is implemented with the `.find()method;` therefore, `$('li.item-ii').find('li')` is equivalent to `$('li', 'li.item-ii')`.

As of jQuery 1.6, we can also filter the selection with a given jQuery collection or element. With the same nested list as above, if we start with:

```
var $allListElements = $('li');
```

And then pass this jQuery object to find:

```
$('li.item-ii').find( $allListElements );
```

This will return a jQuery collection which contains only the list elements that are descendants of item II.

Similarly, an element may also be passed to find:

```
var item1 = $('li.item-1')[0];
$('li.item-ii').find( item1 ).css('background-color', 'red');
```

The result of this call would be a red background on item 1.

Example

Starts with all paragraphs and searches for descendant span elements, same as $("p span")

```
$("p").find("span").css('color','red');
```

Example

A selection using a jQuery collection of all span tags. Only spans within p tags are changed to red while others are left blue.

```
var $spans = $('span');
 $("p").find( $spans ).css('color','red');
```

Example

Add spans around each word then add a hover and italicize words with the letter **t**.

```
var newText = $("p").text().split(" ").join("</span> <span>");
 newText = "<span>" + newText + "</span>";

 $("p").html( newText )
  .find('span')
  .hover(function() {
   $(this).addClass("hilite");
  },
   function() { $(this).removeClass("hilite");
  })
```

```
.end()
 .find(":contains('t')")
 .css({"font-style":"italic", "font-weight":"bolder"});
```

closest(selector)

Get the first element that matches the selector, beginning at the current element and progressing up through the DOM tree.

Arguments

selector - A string containing a selector expression to match elements against.

Given a jQuery object that represents a set of DOM elements, the `.closest()` method searches through these elements and their ancestors in the DOM tree and constructs a new jQuery object from the matching elements. The `.parents()` and `.closest()` methods are similar in that they both traverse up the DOM tree. The differences between the two, though subtle, are significant:

.closest()	.parents()
Begins with the current element	Begins with the parent element
Travels up the DOM tree until it finds a match for the supplied selector	Travels up the DOM tree to the document's root element, adding each ancestor element to a temporary collectio
The returned jQuery object contains zero or one element	The returned jQuery object contains zero, one, or multiple elements

```
<ul id="one" class="level-1">
 <li class="item-i">I</li>
 <li id="ii" class="item-ii">II
 <ul class="level-2">
  <li class="item-a">A</li>
  <li class="item-b">B
   <ul class="level-3">
    <li class="item-1">1</li>
    <li class="item-2">2</li>
    <li class="item-3">3</li>
   </ul>
  </li>
  <li class="item-c">C</li>
 </ul>
 </li>
 <li class="item-iii">III</li>
</ul>
```

Suppose we perform a search for `<ul>` elements starting at item A:

```
$('li.item-a').closest('ul')
 .css('background-color', 'red');
```

This will change the color of the level-2 `<ul>`, since it is the first encountered when traveling up the DOM tree.

Suppose we search for an `<li>` element instead:

```
$('li.item-a').closest('li')
 .css('background-color', 'red');
```

This will change the color of list item A. The `.closest()` method begins its search *with the element itself* before progressing up the DOM tree, and stops when item A matches the selector.

We can pass in a DOM element as the context within which to search for the closest element.

```
var listItemII = document.getElementById('ii');
$('li.item-a').closest('ul', listItemII)
 .css('background-color', 'red');
$('li.item-a').closest('#one', listItemII)
 .css('background-color', 'green');
```

This will change the color of the level-2 <ul>, because it is both the first <ul> ancestor of list item A and a descendant of list item II. It will not change the color of the level-1 <ul>, however, because it is not a descendant of list item II.

Example
Show how event delegation can be done with closest. The closest list element toggles a yellow background when it or its descendent is clicked.

```
$( document ).bind("click", function( e ) {
  $( e.target ).closest("li").toggleClass("hilight");
 });
```

Example
Pass a jQuery object to closest. The closest list element toggles a yellow background when it or its descendent is clicked.

```
var $listElements = $("li").css("color", "blue");
 $( document ).bind("click", function( e ) {
  $( e.target ).closest( $listElements ).toggleClass("hilight");
 });
```

closest(selectors, [context])
Gets an array of all the elements and selectors matched against the current element up through the DOM tree.

Arguments
selectors - An array or string containing a selector expression to match elements against (can also be a jQuery object).
context - A DOM element within which a matching element may be found. If no context is passed in then the context of the jQuery set will be used instead.

This signature (only!) is deprecated as of jQuery 1.7. This method is primarily meant to be used internally or by plugin authors.

Example
Show how event delegation can be done with closest.

```
var close = $("li:first").closest(["ul", "body"]);
 $.each(close, function(i){
 $("li").eq(i).html( this.selector + ": " + this.elem.nodeName );
 });
```

jQuery.map(array, callback(elementOfArray, indexInArray))
Translate all items in an array or object to new array of items.

Arguments
array - The Array to translate.
callback(elementOfArray, indexInArray) - The function to process each item against. The first argument to the function is the array item, the second argument is the index in array The function can return any value. Within the function, `this` refers to the global (window) object.

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 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 - must be converted to actual arrays before being passed to `$.map()`. The jQuery library provides $.makeArray() for such conversions.

```
// The following object masquerades as an array.
var fakeArray = {"length": 1, 0: "Addy", 1: "Subtracty"};

// Therefore, convert it to a real array
```

```
var realArray = $.makeArray( fakeArray )

// Now it can be used reliably with $.map()
$.map( realArray, function(val, i) {
 // do something
});
```

The translation function that is provided to this method is called for each top-level element in the array or object and is passed two arguments: The element's value and its index or key within the array or object.

The function can return:

- the translated value, which will be mapped to the resulting array
- `null`, to remove the item
- an array of values, which will be flattened into the full array

Example

A couple examples of using .map()

```
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(", "));
```

Example

Map the original array to a new one and add 4 to each value.

```
$.map( [0,1,2], function(n){
  return n + 4;
 });
```

Example

Maps the original array to a new one and adds 1 to each value if it is bigger then zero, otherwise it's removed.

```
$.map( [0,1,2], function(n){
  return n > 0 ? n + 1 : null;
 });
```

Example

Map the original array to a new one; each element is added with its original value and the value plus one.

```
$.map( [0,1,2], function(n){
  return [ n, n + 1 ];
 });
```

Example

Map the original object to a new array and double each value.

```
var dimensions = { width: 10, height: 15, length: 20 };
dimensions = $.map( dimensions, function( value, index ) {
 return value * 2;
});
```

Example

Map an object's keys to an array.

```
var dimensions = { width: 10, height: 15, length: 20 },
  keys = $.map( dimensions, function( value, index ) {
   return index;
  });
```

Example
Maps the original array to a new one; each element is squared.

```
$.map( [0,1,2,3], function (a) {
 return a * a;
});
```

Example
Remove items by returning `null` from the function. This removes any numbers less than 50, and the rest are decreased by 45.

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

Example
Augmenting the resulting array by returning an array inside the function.

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

is(selector)
Check the current matched set of elements against a selector, element, or jQuery object and return `true` if at least one of these elements matches the given arguments.

Arguments
selector - A string containing a selector expression to match elements against.

Unlike other filtering methods, `.is()` does not create a new jQuery object. Instead, it allows you to test the contents of a jQuery object without modification. This is often useful inside callbacks, such as event handlers.

Suppose you have a list, with two of its items containing a child element:

```
<ul>
 <li>list <strong>item 1</strong></li>
 <li><span>list item 2</span></li>
 <li>list item 3</li>
</ul>
```

You can attach a click handler to the <ul> element, and then limit the code to be triggered only when a list item itself, not one of its children, is clicked:

```
$("ul").click(function(event) {
 var $target = $(event.target);
 if ( $target.is("li") ) {
  $target.css("background-color", "red");
 }
});
```

Now, when the user clicks on the word "list" in the first item or anywhere in the third item, the clicked list item will be given a red background. However, when the user clicks on item 1 in the first item or anywhere in the second item, nothing will occur, because in those cases the target of the event would be `<strong>` or `<span>`, respectively.

Prior to jQuery 1.7, in selector strings with positional selectors such as `:first`, `:gt()`, or `:even`, the positional filtering is done against the jQuery object passed to `.is()`, *not* against the containing document. So for the HTML shown above, an expression such as

`$("li:first").is("li:last")` returns `true`, but `$("li:first-child").is("li:last-child")` returns `false`. In addition, a bug in Sizzle prevented many positional selectors from working properly. These two factors made positional selectors almost unusable in filters.

Starting with jQuery 1.7, selector strings with positional selectors apply the selector against the document, and then determine whether the first element of the current jQuery set matches any of the resulting elements. So for the HTML shown above, an expression such as `$("li:first").is("li:last")` returns `false`. Note that since positional selectors are jQuery additions and not W3C standard, we recommend using the W3C selectors whenever feasible.

Using a Function

The second form of this method evaluates expressions related to elements based on a function rather than a selector. For each element, if the function returns `true`, `.is()` returns `true` as well. For example, given a somewhat more involved HTML snippet:

```
<ul>
 <li><strong>list</strong> item 1 - one strong tag</li>
 <li><strong>list</strong> item <strong>2</strong> -
  two <span>strong tags</span></li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

You can attach a click handler to every `<li>` that evaluates the number of `<strong>` elements within the clicked `<li>` at that time like so:

```
$("li").click(function() {
 var $li = $(this),
  isWithTwo = $li.is(function() {
   return $('strong', this).length === 2;
  });
 if ( isWithTwo ) {
  $li.css("background-color", "green");
 } else {
  $li.css("background-color", "red");
 }
});
```

Example

Shows a few ways is() can be used inside an event handler.

```
$("div").one('click', function () {
  if ($(this).is(":first-child")) {
   $("p").text("It's the first div.");
  } else if ($(this).is(".blue,.red")) {
   $("p").text("It's a blue or red div.");
  } else if ($(this).is(":contains('Peter')")) {
   $("p").text("It's Peter!");
  } else {
   $("p").html("It's nothing <em>special</em>.");
  }
  $("p").hide().slideDown("slow");
  $(this).css({"border-style": "inset", cursor:"default"});
 });
```

Example

Returns true, because the parent of the input is a form element.

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
 $("div").text("isFormParent = " + isFormParent);
```

Example
Returns false, because the parent of the input is a p element.

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
 $("div").text("isFormParent = " + isFormParent);
```

Example
Checks against an existing collection of alternating list elements. Blue, alternating list elements slide up while others turn red.

```
var $alt = $("#browsers li:nth-child(2n)").css("background", "#00FFFF");
 $('li').click(function() {
  var $li = $(this);
  if ( $li.is( $alt ) ) {
   $li.slideUp();
  } else {
   $li.css("background", "red");
  }
 });
```

Example
An alternate way to achieve the above example using an element rather than a jQuery object. Checks against an existing collection of alternating list elements. Blue, alternating list elements slide up while others turn red.

```
var $alt = $("#browsers li:nth-child(2n)").css("background", "#00FFFF");
 $('li').click(function() {
  if ( $alt.is( this ) ) {
   $(this).slideUp();
  } else {
   $(this).css("background", "red");
  }
 });
```

attr(attributeName)
Get the value of an attribute for the first element in the set of matched elements.

Arguments
attributeName - The name of the attribute to get.

The `.attr()` method gets the attribute value for only the *first* element in the matched set. To get the value for each element individually, use a looping construct such as jQuery's `.each()` or `.map()` method.

As of jQuery 1.6, the `.attr()` method returns `undefined` for attributes that have not been set. In addition, `.attr()` should not be used on plain objects, arrays, the window, or the document. To retrieve and change DOM properties, use the .prop() method.

Using jQuery's `.attr()` method to get the value of an element's attribute has two main benefits:
- **Convenience**: It can be called directly on a jQuery object and chained to other jQuery methods.
- **Cross-browser consistency**: The values of some attributes are reported inconsistently across browsers, and even across versions of a single browser. The `.attr()` method reduces such inconsistencies.

Note: Attribute values are strings with the exception of a few attributes such as value and tabindex.

Example
Find the title attribute of the first in the page.

```
var title = $("em").attr("title");
 $("div").text(title);
```

attr(attributeName, value)
Set one or more attributes for the set of matched elements.

Arguments

attributeName - The name of the attribute to set.
value - A value to set for the attribute.

The `.attr()` method is a convenient way to set the value of attributes-especially when setting multiple attributes or using values returned by a function. Consider the following image:

```
<img id="greatphoto" src="brush-seller.jpg" alt="brush seller" />
```

Setting a simple attribute

To change the `alt` attribute, simply pass the name of the attribute and its new value to the `.attr()` method:

```
$('#greatphoto').attr('alt', 'Beijing Brush Seller');
```

Add an attribute the same way:

```
$('#greatphoto')
.attr('title', 'Photo by Kelly Clark');
```

Setting several attributes at once

To change the `alt` attribute and add the `title` attribute at the same time, pass both sets of names and values into the method at once using a map (JavaScript object literal). Each key-value pair in the map adds or modifies an attribute:

```
$('#greatphoto').attr({
 alt: 'Beijing Brush Seller',
 title: 'photo by Kelly Clark'
});
```

When setting multiple attributes, the quotes around attribute names are optional.

WARNING: When setting the 'class' attribute, you must always use quotes!

Note: jQuery prohibits changing the `type` attribute on an `<input>` or `<button>` element and will throw an error in all browsers. This is because the `type` attribute cannot be changed in Internet Explorer.

Computed attribute values

By using a function to set attributes, you can compute the value based on other properties of the element. For example, to concatenate a new value with an existing value:

```
$('#greatphoto').attr('title', function(i, val) {
 return val + ' - photo by Kelly Clark'
});
```

This use of a function to compute attribute values can be particularly useful when modifying the attributes of multiple elements at once.

Note: If nothing is returned in the setter function (ie. `function(index, attr){}`), or if `undefined` is returned, the current value is not changed. This is useful for selectively setting values only when certain criteria are met.

Example

Set some attributes for all <img>s in the page.

```
$("img").attr({
 src: "/images/hat.gif",
 title: "jQuery",
 alt: "jQuery Logo"
});
$("div").text($("img").attr("alt"));
```

Example
Set the id for divs based on the position in the page.

```
$("div").attr("id", function (arr) {
 return "div-id" + arr;
})
.each(function () {
 $("span", this).html("(ID = '<b>" + this.id + "</b>')");
});
```

Example
Set the src attribute from title attribute on the image.

```
$("img").attr("src", function() {
  return "/images/" + this.title;
});
```

Version 1.7

event.delegateTarget
The element where the currently-called jQuery event handler was attached.

This property is most often useful in delegated events attached by .delegate() or .on(), where the event handler is attached at an ancestor of the element being processed. It can be used, for example, to identify and remove event handlers at the delegation point.

For non-delegated event handlers attached directly to an element, `event.delegateTarget` will always be equal to `event.currentTarget`.

Example
When a button in any box class is clicked, change the box's background color to red.

```
$(".box").on("click", "button", function(event) {
 $(event.delegateTarget).css("background-color", "red");
});
```

callbacks.fired()
Determine if the callbacks have already been called at least once.

Example

Using `callbacks.fired()` to determine if the callbacks in a list have been called at least once:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'
callbacks.fire( 'world '); // outputs: 'foo: world'

// test to establish if the callbacks have been called
```

```
console.log( callbacks.fired() );
```

jQuery.Callbacks(flags)
A multi-purpose callbacks list object that provides a powerful way to manage callback lists.

Arguments
flags - An optional list of space-separated flags that change how the callback list behaves.

The `$.Callbacks()` function is internally used to provide the base functionality behind the jQuery `$.ajax()` and `$.Deferred()` components. It can be used as a similar base to define functionality for new components.

`$.Callbacks()` support a number of methods including `callbacks.add()`,`callbacks.remove()`, `callbacks.fire()` and `callbacks.disable()`.

Getting started

The following are two sample methods named `fn1` and `fn2`:

```
function fn1( value ){
  console.log( value );
}

function fn2( value ){
  fn1("fn2 says:" + value);
  return false;
}
```

These can be added as callbacks to a `$.Callbacks` list and invoked follows:

```
var callbacks = $.Callbacks();
callbacks.add( fn1 );
callbacks.fire( "foo!" ); // outputs: foo!

callbacks.add( fn2 );
callbacks.fire( "bar!" ); // outputs: bar!, fn2 says: bar!
```

The result of this is that it becomes simple to construct complex lists of callbacks where input values can be passed through to as many functions as needed with ease.

Two specific methods were being used above: `.add()` and `.fire()` .add() supports adding new callbacks to the callback list, whilst fire() provides a way to pass arguments to be processed by the callbacks in the same list.

Another method supported by `$.Callbacks` is remove(), which has the ability to remove a particular callback from the callback list. Here"s a practical example of .remove() being used:

```
var callbacks = $.Callbacks();
callbacks.add( fn1 );
callbacks.fire( "foo!" ); // outputs: foo!

callbacks.add( fn2 );
callbacks.fire( "bar!" ); // outputs: bar!, fn2 says: bar!

callbacks.remove(fn2);
callbacks.fire( "foobar" );
```

```
// only outputs foobar, as fn2 has been removed.
```

Supported Flags

The `flags` argument is an optional argument to `$.Callbacks()`, structured as a list of space-separated strings that change how the callback list behaves (eg. `$.Callbacks( 'unique stopOnFalse' )`).

Possible flags:

- `once`: Ensures the callback list can only be fired once (like a Deferred).
- `memory`: Keep track of previous values and will call any callback added after the list has been fired right away with the latest "memorized" values (like a Deferred).
- `unique`: Ensures a callback can only be added once (so there are no duplicates in the list).
- `stopOnFalse`: Interrupts callings when a callback returns false.

By default a callback list will act like an event callback list and can be "fired" multiple times.

For examples of how `flags` should ideally be used, see below:

`$.Callbacks( 'once' )`:

```
var callbacks = $.Callbacks( "once" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
*/
```

`$.Callbacks( 'memory' )`:

```
var callbacks = $.Callbacks( "memory" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
fn2 says:foo
bar
fn2 says:bar
foobar
*/
```

`$.Callbacks( 'unique' )`:

```
var callbacks = $.Callbacks( "unique" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn1 ); // repeat addition
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
bar
fn2 says:bar
foobar
*/
```

`$.Callbacks( 'stopOnFalse' )`:

```
function fn1( value ){
  console.log( value );
  return false;
}

function fn2( value ){
  fn1("fn2 says:" + value);
  return false;
}

var callbacks = $.Callbacks( "stopOnFalse");
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
bar
foobar
*/
```

Because $.Callbacks() supports a list of flags rather than just one, setting several flags has a cumulative effect similar to "&&". This means it's possible to combine flags to create callback lists that are say, both *unique* and *ensure if list was already fired, adding more callbacks will have it called with the latest fired value* (i.e. `$.Callbacks("unique memory")`).

`$.Callbacks( 'unique memory' )`:

```
function fn1( value ){
  console.log( value );
  return false;
}

function fn2( value ){
  fn1("fn2 says:" + value);
```

```
  return false;
}

var callbacks = $.Callbacks( "unique memory" );
callbacks.add( fn1 );
callbacks.fire( "foo" );
callbacks.add( fn1 ); // repeat addition
callbacks.add( fn2 );
callbacks.fire( "bar" );
callbacks.add( fn2 );
callbacks.fire( "baz" );
callbacks.remove( fn2 );
callbacks.fire( "foobar" );

/*
output:
foo
fn2 says:foo
bar
fn2 says:bar
baz
fn2 says:baz
foobar
*/
```

Flag combinations are internally used with `$.Callbacks()` in jQuery for the `.done()` and `.fail()` buckets on a Deferred - both of which use `$.Callbacks('memory once')`.

$.Callbacks methods can also be detached, should there be a need to define short-hand versions for convenience:

```
var callbacks = $.Callbacks(),
  add = callbacks.add,
  remove = callbacks.remove,
  fire = callbacks.fire;

add( fn1 );
fire( "hello world");
remove( fn1 );
```

$.Callbacks, $.Deferred and Pub/Sub

The general idea behind pub/sub (the Observer pattern) is the promotion of loose coupling in applications. Rather than single objects calling on the methods of other objects, an object instead subscribes to a specific task or activity of another object and is notified when it occurs. Observers are also called Subscribers and we refer to the object being observed as the Publisher (or the subject). Publishers notify subscribers when events occur

As a demonstration of the component-creation capabilities of `$.Callbacks()`, it's possible to implement a Pub/Sub system using only callback lists. Using `$.Callbacks` as a topics queue, a system for publishing and subscribing to topics can be implemented as follows:

```
var topics = {};

jQuery.Topic = function( id ) {
  var callbacks,
    method,
    topic = id && topics[ id ];
  if ( !topic ) {
    callbacks = jQuery.Callbacks();
    topic = {
      publish: callbacks.fire,
      subscribe: callbacks.add,
```

```
      unsubscribe: callbacks.remove
    };
    if ( id ) {
      topics[ id ] = topic;
    }
  }
  return topic;
};
```

This can then be used by parts of your application to publish and subscribe to events of interest quite easily:

```
// Subscribers
$.Topic( "mailArrived" ).subscribe( fn1 );
$.Topic( "mailArrived" ).subscribe( fn2 );
$.Topic( "mailSent" ).subscribe( fn1 );

// Publisher
$.Topic( "mailArrived" ).publish( "hello world!" );
$.Topic( "mailSent" ).publish( "woo! mail!" );

// Here, "hello world!" gets pushed to fn1 and fn2
// when the "mailArrived" notification is published
// with "woo! mail!" also being pushed to fn1 when
// the "mailSent" notification is published.

/*
output:
hello world!
fn2 says: hello world!
woo! mail!
*/
```

Whilst this is useful, the implementation can be taken further. Using `$.Deferreds`, it's possible to ensure publishers only publish notifications for subscribers once particular tasks have been completed (resolved). See the below code sample for some further comments on how this could be used in practice:

```
// subscribe to the mailArrived notification
$.Topic( "mailArrived" ).subscribe( fn1 );

// create a new instance of Deferreds
var dfd = $.Deferred();

// define a new topic (without directly publishing)
var topic = $.Topic( "mailArrived" );

// when the deferred has been resolved, publish a
// notification to subscribers
dfd.done( topic.publish );

// Here the Deferred is being resolved with a message
// that will be passed back to subscribers. It's possible to
// easily integrate this into a more complex routine
// (eg. waiting on an ajax call to complete) so that
// messages are only published once the task has actually
// finished.
dfd.resolve( "its been published!" );
```

callbacks.locked()

Determine if the callbacks list has been locked.

Example

Using `callbacks.locked()` to determine the lock-state of a callback list:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value);
}

var callbacks = $.Callbacks();

// add the logging function to the callback list
callbacks.add( foo );

// fire the items on the list, passing an argument
callbacks.fire( 'hello' );
// outputs 'foo: hello'

// lock the callbacks list
callbacks.lock();

// test the lock-state of the list
console.log ( callbacks.locked() ); //true
```

callbacks.empty()

Remove all of the callbacks from a list.

Example

Using `callbacks.empty()` to empty a list of callbacks:

```
// a sample logging function to be added to a callbacks list
var foo = function( value1, value2 ){
  console.log( 'foo:' + value1 + ',' + value2 );
}

// another function to also be added to the list
var bar = function( value1, value2 ){
  console.log( 'bar:' + value1 + ',' + value2 );
}

var callbacks = $.Callbacks();

// add the two functions
callbacks.add( foo );
callbacks.add( bar );

// empty the callbacks list
callbacks.empty();

// check to ensure all callbacks have been removed
console.log( callbacks.has( foo ) ); // false
console.log( callbacks.has( bar ) ); // false
```

callbacks.lock()
Lock a callback list in its current state.

Example

Using `callbacks.lock()` to lock a callback list to avoid further changes being made to the list state:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value);
}

var callbacks = $.Callbacks();

// add the logging function to the callback list
callbacks.add( foo );

// fire the items on the list, passing an argument
callbacks.fire( 'hello' );
// outputs 'foo: hello'

// lock the callbacks list
callbacks.lock();

// try firing the items again
callbacks.fire( 'world' );

// as the list was locked, no items
// were called so 'world' isn't logged
```

callbacks.fire(arguments)
Call all of the callbacks with the given arguments

Arguments
arguments - The argument or list of arguments to pass back to the callback list.

Example

Using `callbacks.fire()` to invoke the callbacks in a list with any arguments that have been passed:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'
callbacks.fire( 'world '); // outputs: 'foo: world'

// add another function to the list
var bar = function( value ){
```

```
  console.log( 'bar:' + value );
}

// add this function to the list
callbacks.add( bar );

// fire the items on the list again
callbacks.fire( 'hello again' );
// outputs:
// 'foo: hello again'
// 'bar: hello again'
```

callbacks.remove(callbacks)

Remove a callback or a collection of callbacks from a callback list.

Arguments

callbacks - A function, or array of functions, that are to be removed from the callback list.

Example

Using `callbacks.remove()` to remove callbacks from a callback list:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( 'foo:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' ); // outputs: 'foo: hello'

// remove 'foo' from the callback list
callbacks.remove( foo );

// fire the items on the list again
callbacks.fire( 'world' );

// nothing output as 'foo' is no longer in the list
```

callbacks.add(callbacks)

Add a callback or a collection of callbacks to a callback list.

Arguments

callbacks - A function, or array of functions, that are to be added to the callback list.

Example

Using `callbacks.add()` to add new callbacks to a callback list:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
```

```
  console.log( 'foo:' + value );
}

// another function to also be added to the list
var bar = function( value ){
  console.log( 'bar:' + value );
}

var callbacks = $.Callbacks();

// add the function 'foo' to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'hello' );
// outputs: 'foo: hello'

// add the function 'bar' to the list
callbacks.add( bar );

// fire the items on the list again
callbacks.fire( 'world' );

// outputs:
// 'foo: world'
// 'bar: world'
```

callbacks.disable()

Disable a callback list from doing anything more.

Example

Using `callbacks.disable()` to disable further calls being made to a callback list:

```
// a sample logging function to be added to a callbacks list
var foo = function( value ){
  console.log( value );
}

var callbacks = $.Callbacks();

// add the above function to the list
callbacks.add( foo );

// fire the items on the list
callbacks.fire( 'foo' ); // outputs: foo

// disable further calls being possible
callbacks.disable();

// attempt to fire with 'foobar' as an argument
callbacks.fire( 'foobar' ); // foobar isn't output
```

callbacks.has(callback)

Determine whether a supplied callback is in a list

Arguments
callback - The callback to search for.

Example

Using `callbacks.has()` to check if a callback list contains a specific callback:

```
// a sample logging function to be added to a callbacks list
var foo = function( value1, value2 ){
  console.log( 'Received:' + value1 + ',' + value2 );
}

// a second function which will not be added to the list
var bar = function( value1, value2 ){
  console.log( 'foobar');
}

var callbacks = $.Callbacks();

// add the log method to the callbacks list
callbacks.add( foo );

// determine which callbacks are in the list

console.log( callbacks.has( foo ) ); // true
console.log( callbacks.has( bar ) ); // false
```

callbacks.fireWith([context], [args])
Call all callbacks in a list with the given context and arguments.

Arguments
context - A reference to the context in which the callbacks in the list should be fired.
args - An argument, or array of arguments, to pass to the callbacks in the list.

Example

Using `callbacks.fireWith()` to fire a list of callbacks with a specific context and an array of arguments:

```
// a sample logging function to be added to a callbacks list
var log = function( value1, value2 ){
  console.log( 'Received:' + value1 + ',' + value2 );
}

var callbacks = $.Callbacks();

// add the log method to the callbacks list
callbacks.add( log );

// fire the callbacks on the list using the context 'window'
// and an arguments array

callbacks.fireWith( window, ['foo','bar']);

// outputs: Received: foo, bar
```

deferred.progress(progressCallbacks)

Add handlers to be called when the Deferred object generates progress notifications.

Arguments

progressCallbacks - A function, or array of functions, that is called when the Deferred generates progress notifications.

The argument can be either a single function or an array of functions. When the Deferred generates progress notifications by calling `notify` or `notifyWith`, the `progressCallbacks` are called. Since `deferred.progress()` returns the Deferred object, other methods of the Deferred object can be chained to this one. When the Deferred is resolved or rejected, progress callbacks will no longer be called. For more information, see the documentation for Deferred object.

deferred.notifyWith(context, [args])

Call the progressCallbacks on a Deferred object with the given context and `args`.

Arguments

context - Context passed to the progressCallbacks as the `this` object.
args - Optional arguments that are passed to the progressCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state or reporting status by returning a restricted Promise object through deferred.promise().

When `deferred.notifyWith` is called, any progressCallbacks added by deferred.then or deferred.progress are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.notifyWith()`. Any calls to `.notifyWith()` after a Deferred is resolved or rejected (or any progressCallbacks added after that) are ignored. For more information, see the documentation for Deferred object.

deferred.notify(args)

Call the progressCallbacks on a Deferred object with the given `args`.

Arguments

args - Optional arguments that are passed to the progressCallbacks.

Normally, only the creator of a Deferred should call this method; you can prevent other code from changing the Deferred's state or reporting status by returning a restricted Promise object through deferred.promise().

When `deferred.notify` is called, any progressCallbacks added by deferred.then or deferred.progress are called. Callbacks are executed in the order they were added. Each callback is passed the `args` from the `.notify()`. Any calls to `.notify()` after a Deferred is resolved or rejected (or any progressCallbacks added after that) are ignored. For more information, see the documentation for Deferred object.

off(events, [selector], [handler(eventObject)])

Remove an event handler.

Arguments

events - One or more space-separated event types and optional namespaces, or just namespaces, such as "click", "keydown.myPlugin", or ".myPlugin".
selector - A selector which should match the one originally passed to `.on()` when attaching event handlers.
handler(eventObject) - A handler function previously attached for the event(s), or the special value `false`.

The `off()` method removes event handlers that were attached with .on(). See the discussion of delegated and directly bound events on that page for more information. Specific event handlers can be removed on elements by providing combinations of event names, namespaces, selectors, or handler function names. **When multiple filtering arguments are given, all of the arguments provided must match for the event handler to be removed.**

If a simple event name such as `"click"` is provided, *all* events of that type (both direct and delegated) are removed from the elements in the jQuery set. When writing code that will be used as a plugin, or simply when working with a large code base, best practice is to attach and remove events using namespaces so that the code will not inadvertently remove event handlers attached by other code. All events of all types in a specific namespace can be removed from an element by providing just a namespace, such as `".myPlugin"`. At minimum, either a namespace or event name must be provided.

To remove specific delegated event handlers, provide a `selector` argument. The selector string must exactly match the one passed to `.on()` when the event handler was attached. To remove all delegated events from an element without removing non-delegated events, use the special value `"**"`.

A handler can also be removed by specifying the function name in the `handler` argument. When jQuery attaches an event handler, it assigns a unique id to the handler function. Handlers proxied by jQuery.proxy() or a similar mechanism will all have the same unique id (the proxy function), so passing proxied handlers to `.off` may remove more handlers than intended. In those situations it is better to attach and remove event handlers using namespaces.

As with `.on()`, you can pass an `events-map` argument instead of specifying the `events` and `handler` as separate arguments. The keys are events and/or namespaces; the values are handler functions or the special value `false`.

Example
Add and remove event handlers on the colored button.

```
function aClick() {
 $("div").show().fadeOut("slow");
}
$("#bind").click(function () {
 $("body").on("click", "#theone", aClick)
  .find("#theone").text("Can Click!");
});
$("#unbind").click(function () {
 $("body").off("click", "#theone", aClick)
  .find("#theone").text("Does nothing...");
});
```

Example
Remove all event handlers from all paragraphs:

```
$("p").off()
```

Example
Remove all delegated click handlers from all paragraphs:

```
$("p").off( "click", "**" )
```

Example
Remove just one previously bound handler by passing it as the third argument:

```
var foo = function () {
 // code to handle some kind of event
};

// ... now foo will be called when paragraphs are clicked ...
$("body").on("click", "p", foo);


// ... foo will no longer be called.
$("body").off("click", "p", foo);
```

Example
Unbind all delegated event handlers by their namespace:

```
var validate = function () {
 // code to validate form entries
};

// delegate events under the ".validator" namespace
$("form").on("click.validator", "button", validate);

$("form").on("keypress.validator", "input[type='text']", validate);

// remove event handlers in the ".validator" namespace

$("form").off(".validator");
```

deferred.state()
Determine the current state of a Deferred object.

The deferred.state() method returns a string representing the current state of the Deferred object. The Deferred object can be in one of three states:

- **"pending"**: The Deferred object is not yet in a completed state (neither "rejected" nor "resolved").
- **"resolved"**: The Deferred object is in the resolved state, meaning that either deferred.resolve() or deferred.resolveWith() has been called for the object and the doneCallbacks have been called (or are in the process of being called).
- **"rejected"**: The Deferred object is in the rejected state, meaning that either deferred.reject() or deferred.rejectWith() has been called for the object and the failCallbacks have been called (or are in the process of being called).

This method is primarily useful for debugging to determine, for example, whether a Deferred has already been resolved even though you are inside code that intended to reject it.

on(events, [selector], [data], handler(eventObject))
Attach an event handler function for one or more events to the selected elements.

Arguments
events - One or more space-separated event types and optional namespaces, such as "click" or "keydown.myPlugin".
selector - A selector string to filter the descendants of the selected elements that trigger the event. If the selector is `null` or omitted, the event is always triggered when it reaches the selected element.
data - Data to be passed to the handler in event.data when an event is triggered.
handler(eventObject) - A function to execute when the event is triggered. The value `false` is also allowed as a shorthand for a function that simply does `return false`.

The `.on()` method attaches event handlers to the currently selected set of elements in the jQuery object. As of jQuery 1.7, the `.on()` method provides all functionality required for attaching event handlers. For help in converting from older jQuery event methods, see .bind(), .delegate(), and .live(). To remove events bound with `.on()`, see .off(). To attach an event that runs only once and then removes itself, see .one()

Event names and namespaces

Any event names can be used for the `events` argument. jQuery will pass through the browser's standard JavaScript event types, calling the `handler` function when the browser generates events due to user actions such as `click`. In addition, the .trigger() method can trigger both standard browser event names and custom event names to call attached handlers.

An event name can be qualified by *event namespaces* that simplify removing or triggering the event. For example, `"click.myPlugin.simple"` defines both the myPlugin and simple namespaces for this particular click event. A click event handler attached via that string could be removed with `.off("click.myPlugin")` or `.off("click.simple")` without disturbing other click handlers attached to the elements. Namespaces are similar to CSS classes in that they are not hierarchical; only one name needs to match. Namespaces beginning with an underscore are reserved for jQuery's use.

In the second form of `.on()`, the `events-map` argument is a JavaScript Object, or "map". The keys are strings in the same form as the `events` argument with space-separated event type names and optional namespaces. The value for each key is a function (or `false` value) that is used as the `handler` instead of the final argument to the method. In other respects, the two forms are identical in their behavior as described below.

Direct and delegated events

The majority of browser events *bubble*, or *propagate*, from the deepest, innermost element (the **event target**) in the document where they occur all the way up to the body and the `document` element. In Internet Explorer 8 and lower, a few events such as `change` and `submit` do not natively bubble but jQuery patches these to bubble and create consistent cross-browser behavior.

If `selector` is omitted or is null, the event handler is referred to as *direct* or *directly-bound*. The handler is called every time an event occurs on the selected elements, whether it occurs directly on the element or bubbles from a descendant (inner) element.

When a `selector` is provided, the event handler is referred to as *delegated*. The handler is not called when the event occurs directly on the bound element, but only for descendants (inner elements) that match the selector. jQuery bubbles the event from the event target up to the element where the handler is attached (i.e., innermost to outermost element) and runs the handler for any elements along that path matching the selector.

Event handlers are bound only to the currently selected elements; they must exist on the page at the time your code makes the call to `.on()`. To ensure the elements are present and can be selected, perform event binding inside a document ready handler for elements that are in the

HTML markup on the page. If new HTML is being injected into the page, select the elements and attach event handlers *after* the new HTML is placed into the page. Or, use delegated events to attach an event handler, as described next.

Delegated events have the advantage that they can process events from *descendant elements* that are added to the document at a later time. By picking an element that is guaranteed to be present at the time the delegated event handler is attached, you can use delegated events to avoid the need to frequently attach and remove event handlers. This element could be the container element of a view in a Model-View-Controller design, for example, or `document` if the event handler wants to monitor all bubbling events in the document. The `document` element is available in the `head` of the document before loading any other HTML, so it is safe to attach events there without waiting for the document to be ready.

In addition to their ability to handle events on descendant elements not yet created, another advantage of delegated events is their potential for much lower overhead when many elements must be monitored. On a data table with 1,000 rows in its `tbody`, this example attaches a handler to 1,000 elements:

```
$("#dataTable tbody tr").on("click", function(event){
alert($(this).text());
});
```

A delegated-events approach attaches an event handler to only one element, the tbody, and the event only needs to bubble up one level (from the clicked `tr` to `tbody`):

```
$("#dataTable tbody").on("click", "tr", function(event){
alert($(this).text());
});
```

The event handler and its environment

The `handler` argument is a function (or the value `false`, see below), and is required unless the `events-map` form is used. You can provide an anonymous handler function at the point of the `.on()` call, as the examples have done above, or declare a named function and pass its name:

```
function notify() { alert("clicked"); }
$("button").on("click", notify);
```

When the browser triggers an event or other JavaScript calls jQuery's `.trigger()` method, jQuery passes the handler an event object it can use to analyze and change the status of the event. This object is a *normalized subset* of data provided by the browser; the browser's unmodified native event object is available in `event.originalEvent`. For example, event.type contains the event name (e.g., "resize") and event.target indicates the deepest (innermost) element where the event occurred.

By default, most events bubble up from the original event target to the `document` element. At each element along the way, jQuery calls any matching event handlers that have been attached. A handler can prevent the event from bubbling further up the document tree (and thus prevent handlers on those elements from running) by calling `event.stopPropagation()`. Any other handlers attached on the current element *will* run however. To prevent that, call `event.stopImmediatePropagation()`. (Event handlers bound to an element are called in the same order that they were bound.)

Similarly, a handler can call `event.preventDefault()` to cancel any default action that the browser may have for this event; for example, the default action on a `click` event is to follow the link. Not all browser events have default actions, and not all default actions can be canceled. See the W3C Events Specification for details.

Returning `false` from an event handler will automatically call `event.stopPropagation()` and `event.preventDefault()`. A `false` value can also be passed for the `handler` as a shorthand for `function(){ return false; }`. So, `$("a.disabled").on("click", false);` attaches an event handler to all links with class "disabled" that prevents them from being followed when they are clicked and also stops the event from bubbling.

When jQuery calls a handler, the `this` keyword is a reference to the element where the event is being delivered; for directly bound events this is the element where the event was attached and for delegated events this is an element matching `selector`. (Note that `this` may not be equal to `event.target` if the event has bubbled from a descendant element.) To create a jQuery object from the element so that it can be used with jQuery methods, use `$(this)`.

Passing data to the handler

If a `data` argument is provided to `.on()` and is not `null` or `undefined`, it is passed to the handler in the event.data property each time an event is triggered. The `data` argument can be any type, but if a string is used the `selector` must either be provided or explicitly passed as `null` so that the data is not mistaken for a selector. Best practice is to use an object (map) so that multiple values can be passed as properties.

As of jQuery 1.4, the same event handler can be bound to an element multiple times. This is especially useful when the `event.data` feature is being used, or when other unique data resides in a closure around the event handler function. For example:

```
function greet(event) { alert("Hello "+event.data.name); }
$("button").on("click", { name: "Karl" }, greet);
$("button").on("click", { name: "Addy" }, greet);
```

The above code will generate two different alerts when the button is clicked.

As an alternative or in addition to the `data` argument provided to the `.on()` method, you can also pass data to an event handler using a second argument to .trigger() or .triggerHandler().

Event performance

In most cases, an event such as `click` occurs infrequently and performance is not a significant concern. However, high frequency events such as `mousemove` or `scroll` can fire dozens of times per second, and in those cases it becomes more important to use events judiciously. Performance can be increased by reducing the amount of work done in the handler itself, caching information needed by the handler rather than recalculating it, or by rate-limiting the number of actual page updates using `setTimeout`.

Attaching many delegated event handlers near the top of the document tree can degrade performance. Each time the event occurs, jQuery must compare all selectors of all attached events of that type to every element in the path from the event target up to the top of the document. For best performance, attach delegated events at a document location as close as possible to the target elements. Avoid excessive use of `document` or `document.body` for delegated events on large documents.

jQuery can process simple selectors of the form `tag#id.class` very quickly when they are used to filter delegated events. So, `"#myForm"`, `"a.external"`, and `"button"` are all fast selectors. Delegated events that use more complex selectors, particularly hierarchical ones, can be several times slower--although they are still fast enough for most applications. Hierarchical selectors can often be avoided simply by attaching the handler to a more appropriate point in the document. For example, instead of `$("body").on("click", "#commentForm .addNew", addComment)` use `$("#commentForm").on("click", ".addNew", addComment)`.

Additional notes

There are shorthand methods for some events such as .click() that can be used to attach or trigger event handlers. For a complete list of shorthand methods, see the events category.

Although strongly discouraged for new code, you may see the pseudo-event-name `"hover"` used as a shorthand for the string `"mouseenter mouseleave"`. It attaches a *single event handler* for those two events, and the handler must examine `event.type` to determine whether the event is `mouseenter` or `mouseleave`. Do not confuse the "hover" pseudo-event-name with the .hover() method, which accepts *one or two* functions.

jQuery's event system requires that a DOM element allow attaching data via a property on the element, so that events can be tracked and delivered. The `object`, `embed`, and `applet` elements cannot attach data, and therefore cannot have jQuery events bound to them.

The `focus` and `blur` events are specified by the W3C to not bubble, but jQuery defines cross-browser `focusin` and `focusout` events that do bubble. When `focus` and `blur` are used to attach delegated event handlers, jQuery maps the names and delivers them as `focusin` and `focusout` respectively. For consistency and clarity, use the bubbling event type names.

jQuery also specifically prevents right and middle clicks from bubbling as they don't occur on the element being clicked. Should working with the middle click be required, the `mousedown` or `mouseup` events should be used with `.on()` instead.

In all browsers, the `load` event does not bubble. In Internet Explorer 8 and lower, the `paste` and `reset` events do not bubble. Such events are not supported for use with delegation, but they *can* be used when the event handler is directly attached to the element generating the event.

The `error` event on the `window` object uses nonstandard arguments and return value conventions, so it is not supported by jQuery. Instead, assign a handler function directly to the `window.onerror` property.

Example
Display a paragraph's text in an alert when it is clicked:

```
$("p").on("click", function(){
alert( $(this).text() );
});
```

Example
Pass data to the event handler, which is specified here by name:

```
function myHandler(event) {
alert(event.data.foo);
}
$("p").on("click", {foo: "bar"}, myHandler)
```

Example
Cancel a form submit action and prevent the event from bubbling up by returning `false`:

```
$("form").on("submit", false)
```

Example
Cancel only the default action by using .preventDefault().

```
$("form").on("submit", function(event) {
 event.preventDefault();
});
```

Example
Stop submit events from bubbling without preventing form submit, using .stopPropagation().

```
$("form").on("submit", function(event) {
 event.stopPropagation();
});
```

Example
Attach and trigger custom (non-browser) events.

```
$("p").on("myCustomEvent", function(e, myName, myValue){
 $(this).text(myName + ", hi there!");
 $("span").stop().css("opacity", 1)
  .text("myName = " + myName)
  .fadeIn(30).fadeOut(1000);
});
$("button").click(function () {
 $("p").trigger("myCustomEvent", [ "John" ]);
});
```

Example
Attach multiple event handlers simultaneously using a map.

```
$("div.test").on({
 click: function(){
  $(this).toggleClass("active");
 },
 mouseenter: function(){
  $(this).addClass("inside");
 },
 mouseleave: function(){
  $(this).removeClass("inside");
 }
```

```
});
```

Example
Click any paragraph to add another after it. Note that .on() allows a click event on any paragraph--even new ones--since the event is handled by the ever-present body element after it bubbles to there.

```
var count = 0;
  $("body").on("click", "p", function(){
   $(this).after("<p>Another paragraph! "+(++count)+"</p>");
  });
```

Example
Display each paragraph's text in an alert box whenever it is clicked:

```
$("body").on("click", "p", function(){
 alert( $(this).text() );
});
```

Example
Cancel a link's default action using the preventDefault method.

```
$("body").on("click", "a", function(event){
 event.preventDefault();
});
```

jQuery.isNumeric(value)
Determines whether its argument is a number.

Arguments
value - The value to be tested.

The `$.isNumeric()` method checks whether its argument represents a numeric value. If so, it returns `true`. Otherwise it returns `false`. The argument can be of any type.

Example
Sample return values of $.isNumeric with various inputs.

```
$.isNumeric("-10"); // true
$.isNumeric(16);   // true
$.isNumeric(0xFF);  // true
$.isNumeric("0xFF"); // true
$.isNumeric("8e5"); // true (exponential notation string)
$.isNumeric(3.1415); // true
$.isNumeric(+10);  // true
$.isNumeric(0144);  // true (octal integer literal)
$.isNumeric("");   // false
$.isNumeric({});   // false (empty object)
$.isNumeric(NaN);  // false
$.isNumeric(null);  // false
$.isNumeric(true);  // false
$.isNumeric(Infinity); // false
$.isNumeric(undefined); // false
```

deferred.pipe([doneFilter], [failFilter])
Utility method to filter and/or chain Deferreds.

Arguments
doneFilter - An optional function that is called when the Deferred is resolved.
failFilter - An optional function that is called when the Deferred is rejected.

The `deferred.pipe()` method returns a new promise that filters the status and values of a deferred through a function. The `doneFilter` and

`failFilter` functions filter the original deferred's resolved / rejected status and values. **As of jQuery 1.7**, the method also accepts a `progressFilter` function to filter any calls to the original deferred's `notify` or `notifyWith` methods. These filter functions can return a new value to be passed along to the piped promise's `done()` or `fail()` callbacks, or they can return another observable object (Deferred, Promise, etc) which will pass its resolved / rejected status and values to the piped promise's callbacks. If the filter function used is `null`, or not specified, the piped promise will be resolved or rejected with the same values as the original.

Example
Filter resolve value:

```
var defer = $.Deferred(),
  filtered = defer.pipe(function( value ) {
   return value * 2;
  });

defer.resolve( 5 );
filtered.done(function( value ) {
 alert( "Value is ( 2*5 = ) 10: " + value );
});
```

Example
Filter reject value:

```
var defer = $.Deferred(),
  filtered = defer.pipe( null, function( value ) {
   return value * 3;
  });

defer.reject( 6 );
filtered.fail(function( value ) {
 alert( "Value is ( 3*6 = ) 18: " + value );
});
```

Example
Chain tasks:

```
var request = $.ajax( url, { dataType: "json" } ),
  chained = request.pipe(function( data ) {
   return $.ajax( url2, { data: { user: data.userId } } );
  });

chained.done(function( data ) {
 // data retrieved from url2 as provided by the first request
});
```

deferred.then(doneCallbacks, failCallbacks)
Add handlers to be called when the Deferred object is resolved or rejected.

Arguments
doneCallbacks - A function, or array of functions, called when the Deferred is resolved.
failCallbacks - A function, or array of functions, called when the Deferred is rejected.

All three arguments (including progressCallbacks, as of jQuery 1.7) can be either a single function or an array of functions. The arguments can also be `null` if no callback of that type is desired. Alternatively, use `.done()`, `.fail()` or `.progress()` to set only one type of callback.

When the Deferred is resolved, the doneCallbacks are called. If the Deferred is instead rejected, the failCallbacks are called. As of jQuery 1.7, the `deferred.notify()` or `deferred.notifyWith()` methods can be called to invoke the progressCallbacks as many times as desired before the Deferred is resolved or rejected.

Callbacks are executed in the order they were added. Since `deferred.then` returns the deferred object, other methods of the deferred object can be chained to this one, including additional `.then()` methods. For more information, see the documentation for Deferred object.

Example

Since the jQuery.get method returns a jqXHR object, which is derived from a Deferred object, we can attach handlers using the `.then` method.

```
$.get("test.php").then(
  function(){ alert("$.get succeeded"); },
  function(){ alert("$.get failed!"); }
);
```

removeData([name])

Remove a previously-stored piece of data.

Arguments

name - A string naming the piece of data to delete.

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. Removing data from jQuery's internal `.data()` cache does not effect any HTML5 `data-` attributes in a document; use `.removeAttr()` to remove those.

When using `.removeData("name")`, jQuery will attempt to locate a `data-` attribute on the element if no property by that name is in the internal data cache. To avoid a re-query of the `data-` attribute, set the name to a value of either `null` or `undefined` (e.g. `.data("name", undefined)`) rather than using `.removeData()`.

As of jQuery 1.7, when called with an array of keys or a string of space-separated keys, `.removeData()` deletes the value of each key in that array or string.

As of jQuery 1.4.3, calling `.removeData()` will cause the value of the property being removed to revert to the value of the data attribute of the same name in the DOM, rather than being set to `undefined`.

Example

Set a data store for 2 names then remove one of them.

```
$("span:eq(0)").text("" + $("div").data("test1"));
  $("div").data("test1", "VALUE-1");
  $("div").data("test2", "VALUE-2");
  $("span:eq(1)").text("" + $("div").data("test1"));
  $("div").removeData("test1");
  $("span:eq(2)").text("" + $("div").data("test1"));
  $("span:eq(3)").text("" + $("div").data("test2"));
```

stop([clearQueue], [jumpToEnd])

Stop the currently-running animation on the matched elements.

Arguments

clearQueue - A Boolean indicating whether to remove queued animation as well. Defaults to `false`.
jumpToEnd - A Boolean indicating whether to complete the current animation immediately. Defaults to `false`.

When `.stop()` is called on an element, the currently-running animation (if any) is immediately stopped. If, for instance, an element is being hidden with `.slideUp()` when `.stop()` is called, the element will now still be displayed, but will be a fraction of its previous height. Callback functions are not called.

If more than one animation method is called on the same element, the later animations are placed in the effects queue for the element. These animations will not begin until the first one completes. When `.stop()` is called, the next animation in the queue begins immediately. If the `clearQueue` parameter is provided with a value of `true`, then the rest of the animations in the queue are removed and never run.

If the `jumpToEnd` argument is provided with a value of `true`, the current animation stops, but the element is immediately given its target values for each CSS property. In our above `.slideUp()` example, the element would be immediately hidden. The callback function is then immediately called, if provided.

As of jQuery 1.7, if the first argument is provided as a string, only the animations in the queue represented by that string will be stopped.

The usefulness of the `.stop()` method is evident when we need to animate an element on `mouseenter` and `mouseleave`:

```
<div id="hoverme">
 Hover me
 <img id="hoverme" src="book.png" alt="" width="100" height="123" />
</div>
```

We can create a nice fade effect without the common problem of multiple queued animations by adding `.stop(true, true)` to the chain:

```
$('#hoverme-stop-2').hover(function() {
 $(this).find('img').stop(true, true).fadeOut();
}, function() {
 $(this).find('img').stop(true, true).fadeIn();
});
```

Toggling Animations

As of jQuery 1.7, stopping a toggled animation prematurely with `.stop()` will trigger jQuery's internal effects tracking. In previous versions, calling the `.stop()` method before a toggled animation was completed would cause the animation to lose track of its state (if jumpToEnd was false). Any subsequent animations would start at a new "half-way" state, sometimes resulting in the element disappearing. To observe the new behavior, see the final example below.

Animations may be stopped globally by setting the property `$.fx.off` to `true`. When this is done, all animation methods will immediately set elements to their final state when called, rather than displaying an effect.

Example

Click the Go button once to start the animation, then click the STOP button to stop it where it's currently positioned. Another option is to click several buttons to queue them up and see that stop just kills the currently playing one.

```
/* Start animation */
$("#go").click(function(){
$(".block").animate({left: '+=100px'}, 2000);
});

/* Stop animation when button is clicked */
$("#stop").click(function(){
$(".block").stop();
});

/* Start animation in the opposite direction */
$("#back").click(function(){
$(".block").animate({left: '-=100px'}, 2000);
});
```

Example

Click the slideToggle button to start the animation, then click again before the animation is completed. The animation will toggle the other direction from the saved starting point.

```
var $block = $('.block');
/* Toggle a sliding animation animation */
$('#toggle').on('click', function() {
  $block.stop().slideToggle(1000);
});
```

is(selector)

Check the current matched set of elements against a selector, element, or jQuery object and return `true` if at least one of these elements matches the given arguments.

Arguments

selector - A string containing a selector expression to match elements against.

Unlike other filtering methods, `.is()` does not create a new jQuery object. Instead, it allows you to test the contents of a jQuery object without modification. This is often useful inside callbacks, such as event handlers.

Suppose you have a list, with two of its items containing a child element:

```
<ul>
 <li>list <strong>item 1</strong></li>
 <li><span>list item 2</span></li>
 <li>list item 3</li>
</ul>
```

You can attach a click handler to the <ul> element, and then limit the code to be triggered only when a list item itself, not one of its children, is clicked:

```
$("ul").click(function(event) {
 var $target = $(event.target);
 if ( $target.is("li") ) {
  $target.css("background-color", "red");
 }
});
```

Now, when the user clicks on the word "list" in the first item or anywhere in the third item, the clicked list item will be given a red background. However, when the user clicks on item 1 in the first item or anywhere in the second item, nothing will occur, because in those cases the target of the event would be `<strong>` or `<span>`, respectively.

Prior to jQuery 1.7, in selector strings with positional selectors such as `:first`, `:gt()`, or `:even`, the positional filtering is done against the jQuery object passed to `.is()`, *not* against the containing document. So for the HTML shown above, an expression such as `$("li:first").is("li:last")` returns `true`, but `$("li:first-child").is("li:last-child")` returns `false`. In addition, a bug in Sizzle prevented many positional selectors from working properly. These two factors made positional selectors almost unusable in filters.

Starting with jQuery 1.7, selector strings with positional selectors apply the selector against the document, and then determine whether the first element of the current jQuery set matches any of the resulting elements. So for the HTML shown above, an expression such as `$("li:first").is("li:last")` returns `false`. Note that since positional selectors are jQuery additions and not W3C standard, we recommend using the W3C selectors whenever feasible.

Using a Function

The second form of this method evaluates expressions related to elements based on a function rather than a selector. For each element, if the function returns `true`, `.is()` returns `true` as well. For example, given a somewhat more involved HTML snippet:

```
<ul>
 <li><strong>list</strong> item 1 - one strong tag</li>
 <li><strong>list</strong> item <strong>2</strong> -
  two <span>strong tags</span></li>
 <li>list item 3</li>
 <li>list item 4</li>
 <li>list item 5</li>
</ul>
```

You can attach a click handler to every `<li>` that evaluates the number of `<strong>` elements within the clicked `<li>` at that time like so:

```
$("li").click(function() {
 var $li = $(this),
  isWithTwo = $li.is(function() {
   return $('strong', this).length === 2;
  });
 if ( isWithTwo ) {
  $li.css("background-color", "green");
 } else {
  $li.css("background-color", "red");
 }
```

```
});
```

Example
Shows a few ways is() can be used inside an event handler.

```
$("div").one('click', function () {
  if ($(this).is(":first-child")) {
   $("p").text("It's the first div.");
  } else if ($(this).is(".blue,.red")) {
   $("p").text("It's a blue or red div.");
  } else if ($(this).is(":contains('Peter')")) {
   $("p").text("It's Peter!");
  } else {
   $("p").html("It's nothing <em>special</em>.");
  }
  $("p").hide().slideDown("slow");
  $(this).css({"border-style": "inset", cursor:"default"});
 });
```

Example
Returns true, because the parent of the input is a form element.

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
 $("div").text("isFormParent = " + isFormParent);
```

Example
Returns false, because the parent of the input is a p element.

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
 $("div").text("isFormParent = " + isFormParent);
```

Example
Checks against an existing collection of alternating list elements. Blue, alternating list elements slide up while others turn red.

```
var $alt = $("#browsers li:nth-child(2n)").css("background", "#00FFFF");
 $('li').click(function() {
  var $li = $(this);
  if ( $li.is( $alt ) ) {
   $li.slideUp();
  } else {
   $li.css("background", "red");
  }
 });
```

Example
An alternate way to achieve the above example using an element rather than a jQuery object. Checks against an existing collection of alternating list elements. Blue, alternating list elements slide up while others turn red.

```
var $alt = $("#browsers li:nth-child(2n)").css("background", "#00FFFF");
 $('li').click(function() {
  if ( $alt.is( this ) ) {
   $(this).slideUp();
  } else {
   $(this).css("background", "red");
  }
 });
```

removeAttr(attributeName)
Remove an attribute from each element in the set of matched elements.

Arguments

attributeName - An attribute to remove; as of version 1.7, it can be a space-separated list of attributes.

The `.removeAttr()` method uses the JavaScript `removeAttribute()` function, but it has the advantage of being able to be called directly on a jQuery object and it accounts for different attribute naming across browsers.

Note: Removing an inline `onclick` event handler using `.removeAttr()` doesn't achieve the desired effect in Internet Explorer 6, 7, or 8. To avoid potential problems, use `.prop()` instead:

```
$element.prop("onclick", null);
console.log("onclick property: ", $element[0].onclick);
```

Example

Clicking the button enables the input next to it.

```
(function() {
 var inputTitle = $("input").attr("title");
 $("button").click(function () {
  var input = $(this).next();

  if ( input.attr("title") == inputTitle ) {
   input.removeAttr("title")
  } else {
   input.attr("title", inputTitle);
  }

  $("#log").html( "input title is now " + input.attr("title") );
 });
})();
```

Index