jQuery Core 3.0 Upgrade Guide

jQuery Core 3.0 Upgrade Guide

Overview

With the major version of 3.0, the jQuery Core team has taken the opportunity to make changes to clean up the API and fix bugs that may prove to be breaking changes for some code. This includes the removal of previously deprecated public APIs, changes to or removal of undocumented APIs, and changes to the documented or undocumented behavior of existing APIs for specific inputs.

Browser Support

As of jQuery 3.0, the following browsers are supported:

Internet Explorer: 9+

Chrome, Edge, Firefox, Safari: Current and Current - 1

Opera: Current

Safari Mobile iOS: 7+

Android 4.0+

jQuery team policy is to only change browser support on major-version updates, so this list will apply until at least jQuery 4 arrives.

jQuery Migrate Plugin

As with the major changes made in jQuery 1.9/2.0, we have created a new version of the jQuery Migrate Plugin to simplify migration of older code to version 3.0. We strongly recommend that you use this plugin as an upgrading tool, it will give specific advice about most of the major changes that may affect your code.

Version 3.0 of the jQuery Migrate Plugin *does not* warn about or restore behaviors that were removed in previous major version changes such as jQuery 1.9/2.0. Use the following steps to upgrade from a version of jQuery older than 1.11.0 or 2.1.0 to this new version 3.0:

1. Upgrade the version of jQuery on the page to the latest 1.x or 2.x version (currently 1.12.3 or 2.2.3).
2. Add the uncompressed jQuery Migrate 1.x Plugin to the page.
3. Optional but recommended, update any plugins in use since later versions are usually the most compatible with recent versions of jQuery.

Test the page and resolve any warnings that appear on the console, using the JQMIGRATE 1.x warning documentationas a guide.

Remove the jQuery Migrate 1.x plugin and ensure that the updated jQuery code on the page continues to work properly with only the latest jQuery 1.x/2.x in use.

Upgrade the version of jQuery on the page to the latest 3.0 version (currently 3.0.0) and add the uncompressed jQuery Migrate 3.x plugin to the page.

Test the page and resolve any warnings that appear on the console, using the JQMIGRATE 3.x warning documentationas a guide. Report any bugs in third-party plugins to the plugin author.

Remove the jQuery Migrate 3.x plugin and ensure that the page continues to work properly with only the latest jQuery 3.x in use.

Running both Migrate 1.x and Migrate 3.x simultaneously on the same page is not supported.

The uncompressed development version of the Migrate plugin includes console log output to warn when specific deprecated and/or removed features are being used. This makes it valuable as a migration debugging tool for finding and remediating issues in existing jQuery code and plugins.

The compressed version of the Migrate plugin does not generate any warnings, although it does issue a solitary console message that it has been installed to simplify debugging. Migrate can be used on production sites when jQuery 3.0 or higher is desired but older incompatible jQuery code or plugins must also be used. Ideally this is only used as a short-term solution, since restoring old behavior may cause conflicts with new jQuery code that expects the new behavior.

Summary of Important Changes

With a library as widely used as jQuery, it is often difficult for the team to know which changes may impact developers the most before a release occurs. Despite the length of this list, we believe that the majority are edge cases. Many jQuery projects should be able to run version 3.0 with only minor changes if any.

Changes are listed by their component category, and prefixed with a description to help you understand its impact:

Breaking change: This change *may* affect existing code, since it changes the API surface in some way. Most of the time the impacts are only for specific edge cases as noted.

Feature: The change is an API addition and should not affect existing code in most cases. However, there is the possibility that new features can interact negatively with existing code.

Deprecated: This feature or API is still present in jQuery 3.0, but its use is discouraged. It may be removed in a future major-version update.

Remember that the jQuery Migrate plugin described above can detect and warn about many of these changes so that they can be fixed in your code.

For a complete and detailed list of all code changes, see the 3.0 milestone in the jQuery Core issue tracker or the version diff.

Ajax

Breaking change: Special-case Deferred methods removed from jQuery.ajax

The `jqXHR` object returned from `jQuery.ajax()` is a jQuery `Deferred` and has historically had three extra methods with names matching the arguments object of `success`, `error`, and `complete`. This often confused people who did not realize that the returned object should be treated like a `Deferred`. As of jQuery 3.0 these methods have been removed. As replacements, use the `Deferred` standard methods of `done`, `fail`, and `always`, or use the new `then` and `catch` methods for Promises/A+ compliance.

Note that this does not have any impact at all on the ajax callbacks of the same name passed through the `options` object, which continue to exist and are not deprecated. This only affects the jqXHR methods.

https://github.com/jquery/jquery/issues/2084

Breaking change: Cross-domain script requests must be declared

When making a request via `jQuery.ajax()` or `jQuery.get()` for a script on a domain other than the one that hosts the document, you must now explicitly specify `dataType: "script"` in the options. This prevents the possibility of an attack where the remote site delivers non-script content but later decides to serve a script that has malicious intent. Since `jQuery.getScript()` explicitly sets `dataType: "script"` it is unaffected by this change.

Breaking change: Hash in a URL is preserved in a `jQuery.ajax()` call

The `jQuery.ajax()` method no longer strips off the hash in the URL if it is provided, and sends the full URL to the transport (xhr, script, jsonp, or custom transport). If the server at the other end of the connection cannot deal with a hash on a URL, strip it off before sending the request.

https://github.com/jquery/jquery/issues/1732

Feature: New signature for jQuery.get() AND jQuery.post()

jQuery 3 adds a new signature for the `jQuery.get()` and the `jQuery.post()` functions by adding a `settings` parameter. It's an object that can possess many properties and its the same object that you can provide to `jQuery.ajax()`.

https://github.com/jquery/jquery/issues/1986

Attributes

Breaking change: `.removeAttr()` no longer sets properties to false

Prior to jQuery 3.0, using `.removeAttr()` on a boolean attribute such as `checked`, `selected`, or `readonly` would also set the corresponding named *property* to `false`. This behavior was required for ancient versions of Internet Explorer but is not correct for modern browsers because the attribute represents the initial value and the property represents the current (dynamic) value.

It is almost always a mistake to use `.removeAttr( "checked" )` on a DOM element. The only time it might be useful is if the DOM is later going to be serialized back to an HTML string. In all other cases, `.prop( "checked", false )` should be used instead.

https://github.com/jquery/jquery/issues/1759
https://github.com/jquery/jquery/issues/2913

Breaking change: select-multiple with nothing selected returns an empty array

Before jQuery 3.0, calling `.val()` on a `<select multiple>` element with no elements selected returned `null`. This was inconvenient since if at least one value was selected the return value would be an array. Also, if all options are disabled jQuery already returned an empty array. To improve consistency, the nothing-selected case now returns an empty array.

https://github.com/jquery/jquery/issues/2562

Feature: SVG documents support class operations

SVG has never been fully supported by jQuery and this hasn't changed in jQuery 3. Nonetheless, many jQuery methods work with SVG documents as well. As of jQuery 3 the methods that manipulate class names, such as `.addClass()` and `.hasClass()`, support SVG.

https://github.com/jquery/jquery/issues/2199

Deprecated: `.toggleClass()` with no arguments and `.toggleClass( Boolean )`

Although a signature for these two cases was documented, its actual behavior was never fully defined. This undefined behavior is now deprecated so it will not be documented. You may find you were accidentally using the functionality because `.toggleClass( undefined )` behaves the same as a call with no arguments, even though it was invalid input.

Callbacks

Feature: Locking a Callback prevents only future list execution

If a Callback object has a handler function that calls the `.lock()` method after being `.fire()`d, it only prevents future execution of the callback list and does not immediately abort execution of the current list. To stop the current execution, use the `stopOnFalse` option.

https://github.com/jquery/jquery/issues/1990

Core

Breaking change: jQuery 3.0 runs in Strict Mode

Now that most of the browsers supported by jQuery 3.0 have `"use strict"`, jQuery is being built with this directive. Your code is not required to run in Strict Mode, so most existing code should not require any changes. The one case we encountered three years ago was that ASP.NET 4.0 used `arguments.caller.callee` to attempt tracing through call stacks in its `__doPostBack()` method. If you are still using a version of ASP.NET that still does this, keep using jQuery 2.x or earlier. Modern browsers support stack traces via `error.stack` so it should not ever be necessary to examine `arguments.caller.callee`.

https://github.com/jquery/jquery/pull/3061
https://bugs.jquery.com/ticket/13335

Breaking change: document-ready handlers are now asynchronous

The document-ready processing in jQuery has been powered by the `jQuery.Deferred` implementation since jQuery 1.6. As part of jQuery 3.0's alignment with the Promises/A+ standard, document-ready handlers are called asynchronously even if the document is currently ready at the point where the handler is added. This provides a consistent code execution order that is independent of whether the document is ready or not. For example, consider this code:

```
$(function(){
   console.log("ready");
});
console.log("outside ready");
```

In jQuery 3.0 this will always log "outside ready" followed by "ready" regardless of whether the document is ready at the point of execution. Earlier versions may log the messages in either order.

Since handlers now execute independently of each other, an exception or failure in one document-ready handler no longer prevents other document-ready handlers from running.

In custom builds where the `deferred` module is excluded, an alternative implementation of the document-ready code is used. This implementation supports the `jQuery.ready` promise but is not a `jQuery.Deferred` object so it should only be used through `jQuery.when()`.

https://github.com/jquery/jquery/issues/1823
https://github.com/jquery/jquery/pull/2891

Breaking change: jQuery.isNumeric() and custom `.toString()`

The `jQuery.isNumeric()` method is intended to be used with primitive numbers and strings that can be coerced to finite numbers. In particular, it no longer tries to obtain numbers from objects that have a `.toString()` method. Users needing

specialized checks for other numerics should create their own validation functions.

https://github.com/jquery/jquery/issues/2662

**Breaking change:
Deprecated `.context` and `.selector` properties removed**

These properties were deprecated in jQuery 1.9, as they were only used for the obsolete `.live()` method and have never accurately represented the context or selector for the current collection.

https://github.com/jquery/jquery/issues/1908

Breaking change: Deprecated `.size()` removed

`.size()` is deprecated as of jQuery 1.8 and removed in jQuery 3.0 in favor of the `.length` property.

Breaking change: Undocumented internal methods no longer exposed

Version 3.0 removes several methods from view that were intended to be private and were never documented:

> jQuery.swap
>
> jQuery.buildFragment
>
> jQuery.domManip

https://github.com/jquery/jquery/issues/2224
https://github.com/jquery/jquery/issues/2225

Breaking change: Return values on empty sets are `undefined`

With few exceptions, any value-returning jQuery methods should return `undefined` on an empty jQuery collection in keeping with our API guidelines. The following APIs were changed to conform to this rule:

> Dimensional methods: `.width()`, `.height()`, `.innerWidth()`, `.innerHeight()`, `.outerWidth()`, and `.outerHeight()`
>
> Offset methods: `.offsetTop()` and `.offsetLeft()`

Previously, these methods returned `null` instead of `undefined` for an empty collection.

https://github.com/jquery/jquery/issues/2319

Feature: `for...of` loops can be used on jQuery collections

jQuery 3.0 supports the `for...of` loop introduced in ES2015. It allows looping over iterable objects including `Array`, `Map`, and `Set`. When using this loop, the value obtained is a DOM element of the jQuery collection, one at a time. Note that you will need to be using an environment that supports ES2015 or a transpiler such as Babel to use `for...of`. Here is an example:

```
var elems = $(".someclass");

// Classic jQuery way
$.each(function(i, elem) {
  // work with elem (or "this" object)
});

// Prettier ES2015 way
for ( let elem of elems ) {
  // work with elem
}
```

https://github.com/jquery/jquery/issues/1693

Feature: `jQuery.ready` promise is formally supported

`jQuery.ready` has been consumable as a promise-like object ("thenable" in Promise terms) since jQuery version 1.8. As of jQuery 3.0 this object is documented as supported via `jQuery.when` or the native `Promise.resolve()`. No code should make assumptions about whether this object is a jQuery `Deferred` or some other type of promise object such as a native Promise. Typical usage might look like this:

```
$.when( $.ready, $.getScript("optional.js
   // the document is ready and optional
}).catch( function() {
   // an error occurred
});
```

https://github.com/jquery/api.jquery.com/pull/530

Deprecated: `jQuery.unique()`, renamed to `jQuery.uniqueSort()`

The `jQuery.unique()` method has been renamed to `jQuery.uniqueSort()` to make its behavior easier to understand. There is no change to functionality here, only a rename.

Deprecated: `jQuery.parseJSON()`

Since all the browsers supported by jQuery 3.0 support the native `JSON.parse()` method, we are deprecating `jQuery.parseJSON()`.

https://github.com/jquery/jquery/issues/2800

Deprecated: document-ready handlers other than `jQuery(function)`

Due to historical compatibility issues there are a multitude of ways to set a document ready handler. All of the following are equivalent and call the function `fn` when the document is ready:

```
$(fn);
$().ready(fn);
$(document).ready(fn);
$("selector").ready(fn);
```

As of jQuery 3.0 the recommended way to add a ready handler is the first method, `$(fn)`. As noted in the Event section, the `$(document).on("ready", fn)` event form has slightly different semantics and was removed in jQuery 3.0.

Data

Breaking change: .data() names containing dashes

As of jQuery 3.0, all data names are stored in jQuery's internal data object in camelCase (e.g., `clickCount`), rather than kebab-case (e.g. `click-count`). This is consistent with the way that standard DOM turns dashed names into camel case for JavaScript names in CSS and data properties.

In general, kebab case still works in jQuery 3.0 when setting or getting a specific data item, e.g. `.data("right-aligned")`, but if you retrieve the internal data object it will now have the data item in camel case (`rightAligned`). The main difference in 3.0 is when you use kebab case names directly on the data object instead of using the `.data()` API to get or set them.

For example:

```
var $div = $("<div />");
$div.data("clickCount", 2);
$div.data("clickCount"); // 2
$div.data("click-count", 3);
$div.data("clickCount"); // 3
$div.data("click-count"); // 3

var allData = $div.data();
allData.clickCount; // 3
allData["click-count"]; // undefined
allData["click-count"] = 14;
$div.data("click-count"); // 3, NOT 14 a
allData.clickCount; // 3
allData["click-count"]; // 14
```

https://github.com/jquery/jquery/issues/2070
https://github.com/jquery/jquery/issues/2257
https://github.com/jquery/jquery/issues/1751

Deferred

Breaking change and Feature: jQuery.Deferred is now Promises/A+ compatible

Deferreds have been updated for compatibility with Promises/A+ and ES2015 (a.k.a ES6) Promise, a change with significant consequences.

https://github.com/jquery/jquery/issues/1722
https://github.com/jquery/jquery/issues/2102

Resolution

`.resolve`, `.reject`, and `.notify` now set `undefined` context instead of using the promise of the Deferred object with which they are associated. To set an explicit context, use `.resolveWith`, `.rejectWith`, and `.notifyWith`.

https://github.com/jquery/jquery/issues/3060

Callback exit

Major changes have been made to the `.then()` method. In particular, any exception thrown within a `.then()` callback is now caught and converted into a rejection value, and any non-thenable value returned from a *rejection* handler becomes a fulfillment value. We **strongly recommend** that you add a `.catch()` method (new in 3.0) to the end of your promise chain to avoid difficult debugging issues. The most likely place you may encounter this new behavior is when using the Deferreds that are produced by `jQuery.ajax()`, since the `jQXHR` object returned is a superset of `jQuery.Deferred`.

In jQuery 1.x and 2.x, an uncaught exception inside a callback function halts code execution. The thrown exception bubbles up until it is caught inside a try/catch or reaches `window` and triggers `window.onerror`.

For example, consider this code using the new standard Promises/A+ behavior:

```
$.ajax("/status")
   .then(function(data) {
      whoops();
      // console shows "jQuery.Deferred
      // no further code executes in thi
   })
   .catch(function(arg) {
      // this code executes after the er
      // arg is an Error object, "whoops
   });
```

Compare that to the old-style Deferred methods:

```
$.ajax("/status")
   .done(function(data) {
      whoops();
      // console shows: "whoops is not a
      // no further code executes in this
   })
   .fail(function(arg) {
      // this code does not execute since
   });
```

Note that jQuery logs a message to the console when it is inside a Deferred and a JavaScript exception occurs. These messages take the form `jQuery.Deferred exception: (error message)`. If you do not want any console output on these exceptions, set `jQuery.Deferred.exceptionHook` to `undefined`. If you need further help in finding errors reported this way, use the jquery-deferred-reporter plugin during development to obtain stack traces.

https://github.com/jquery/jquery/issues/2736

Callback invocation

The Promises/A+ spec says that promises are always resolved with a *single value* and handlers are invoked without a `this` context, while jQuery Deferreds sometimes pass context and/or multiple values to their handlers. In most cases, though, the first argument is the most important of these values. If you mix other Promises/A+ implementations with jQuery's, you may only receive a single argument in your handler. To maintain full compatibility for existing code, use only jQuery Deferreds and switch to the older `.done()` and `.fail()` methods which retain all the backward-compatible behavior:

```
1  // Typical old uses of .then() that are
2  $.ajax("url").then(
3      // success
4      function( data, textStatus, jqXHR )
5      // error
6      function( jqXHR, textStatus, errorTh
7  );
8
9  // Rewrite to this in order to maintain
10 $.ajax("url")
11     // success
12     .done(function( data, textStatus, jq
13     // error
14     .fail(function( jqXHR, textStatus, e
```

Another behavior change required for Promises/A+ compliance is that Deferred `.then()` callbacks are *always* called asynchronously. Previously, if a `.then()` callback was added to a Deferred that was already resolved or rejected, the callback would run immediately and synchronously.

Backwards compatibility

Deferred methods such as `.done()`, `.fail()`, and `.pipe()` retain their old behavior and so are not Promises/A+ compliant. If you require synchronous resolution, do not want exceptions converted to rejection values or rejection callback returns converted to fulfillment values, or want thrown errors to bubble out of the function where they occur, you can use these methods instead of `.then()` and `.catch()`.

Breaking change and Feature: jQuery.when() arguments

`jQuery.when` now interprets any input argument with a `then` method as a Promise-compatible "thenable". This allows a

much broader range of inputs, including native ES6 Promises and Bluebird promises.

Further, a more clear distinction is now recognized between multi-argument calls to `jQuery.when` and single- or no-argument calls. Multi-argument calls behave similarly to `Promise.all`, aggregating fulfillment values into a fulfillment array (with jQuery-specific enhancements of also aggregating fulfillment contexts and supporting multi-valued fulfillments), or alternatively rejecting with the first rejection value. Single- and no-argument calls behave similarly to `Promise.resolve`, returning a Deferred that resolves identically to thenable or Promise-like input, or fulfills with its non-Promise input (as appropriate). As of jQuery 3.0, both of these return a new Deferred (previous versions did not create a new Deferred when called with a single Deferred input).

https://github.com/jquery/jquery/issues/2018
https://github.com/jquery/jquery/issues/2546
https://github.com/jquery/jquery/issues/3029

Breaking change: jQuery.when() progress notifications

As of jQuery 3.0, the `jQuery.when()` method no longer passes along progress notifications from input Deferreds to the output Deferred. Progress messages are not part of the Promises/A+ specification, and the behavior of progress notifications in `jQuery.when()` were not previously documented in the jQuery API.

https://github.com/jquery/jquery/issues/2710

Dimensions

Breaking change: .width(), .height(), .css("width"), and .css("height") can return non-integer values

Before version 3.0, jQuery used the DOM `offsetWidth` and `offsetHeight` properties to determine the dimensions of an element, and these properties always return integers. With jQuery 3.0 we get more precise values via the DOM `getBoundingClientRect` API, and these may may not be integers. If your code always expects integers for dimensions, it may need to be adjusted to deal with this extra precision.

https://github.com/jquery/jquery/issues/1724

Breaking change: `.outerWidth()` or `.outerHeight()` on `window` includes scrollbar width/height

Calls to `$(window).width()` return the "content width" which excludes any scrollbars that the browser has added if the content exceeds the height of the element. This is different from the width that CSS uses for media queries, which includes the width of the scrollbars. In order to provide a measure that is equivalent to the CSS media query concept of width, the `$(window).outerWidth()` method now returns the width including scrollbar width. This is equivalent to the DOM property `window.innerWidth`. The same applies for `.outerHeight()`.

https://github.com/jquery/jquery/issues/1729

Effects

Breaking change: .show(), .hide(), and .toggle() methods now respect more stylesheet changes

The code that jQuery uses to show and hide elements has been updated to focus on inline rather than computed styles, respecting stylesheet `display` values whenever possible for increased compatibility with responsive design techniques (in which active stylesheet rules can dynamically change upon device reorientation/window resize/etc.). As a result, disconnected elements are no longer considered hidden unless they have inline `display: none`, and therefore `.toggle()` will no longer differentiate them from connected elements as of jQuery 3.0.

Further, while `.show()` and similar calls will continue to force visibility of elements that are hidden by stylesheet rules, supporting this functionality slows down all show/hide operations and its use is not recommended. The determination of which display value to set in such cases has also been simplified, defaulting to "block" when body-level rules hide elements by type.

Any code expecting hidden elements to be reshown with their previous *computed* display styles, or disconnected elements to treated as hidden, should be reviewed. The team created a table of all the possibilities related to the display state and show/hide actions in order to minimize the setting of non-empty inline styles.

Feature: Animations now use requestAnimationFrame

On platforms that support the `requestAnimationFrame` API, which is pretty much everywhere but IE9, jQuery will now use that API when performing animations. This should result in animations that are smoother and use less CPU time—and save battery as well on mobile devices.

jQuery tried using requestAnimationFrame a few years back but there were serious compatibility issues with existing code so we had to back it out. We think we've beaten most of those issues by suspending animations while a browser tab is out of view. Still, any code that depends on animations to always run in nearly real-time is making an unrealistic assumption.

Deprecated: jQuery.fx.interval

Now that `requestAnimationFrame` is being used for animations, the `jQuery.fx.interval` property is ignored on most browsers. It is still present in jQuery 3.0 and used in browsers such as IE9, but will be removed in a future major-point release.

Deprecated: Additional easing function parameters

The easing functions called by `.animate()` are passed single argument, the percentage of completion. Some older code assumes that it is passed additional arguments derived from the percentage. These other arguments may not be present in a future major version update.

Example of an old easing method:

```
$.easing.easeInOutSine =  function (x, t,
  return -c/2 * (Math.cos(Math.PI*t/d) -
};
```

The same easing method rewritten:

```
$.easing.easeInOutSine =  function (x) {
  return -0.5*(Math.cos(Math.PI*x) - 1);
};
```

https://github.com/jquery/api.jquery.com/issues/912

Event

Breaking change: .load(), .unload(), and .error() removed

These methods are shortcuts for event operations, but had several API limitations. The event `.load()` method conflicted with the ajax `.load()` method. The `.error()` method could not be used with `window.onerror` because of the way the DOM method is defined. If you need to attach events by these names, use the `.on()` method, e.g.
change `$("img").load(fn)` to `$("img").on("load", fn)`.

https://github.com/jquery/jquery/issues/2286

Breaking change: `.on("ready", fn)` removed

jQuery no longer supports a synthetic event named `"ready"` that can be used with the event functions. This event was error-prone and deprecated in jQuery 1.8 because it would only call the callback if it was attached before the document was ready. Replace any uses with `$(fn)` instead, which works reliably.

https://github.com/jquery/jquery/issues/2264

Breaking change: `event.pageX` and `event.pageY` normalization removed

All the browsers officially supported by jQuery 3.0 provide the pageX and pageY properties in their events, so the jQuery code to calculate these values from other event properties has been removed. This change should not affect mainstream browsers, but there may be obscure environments where these properties are not present. If you find one, please open a ticket.

https://github.com/jquery/jquery/issues/3092

Breaking change: `jQuery.event.props` and `jQuery.event.fixHooks` removed

jQuery's event handling performance increased thanks to a reorganization of event property management. The main improvement is that jQuery now only calculates or copies a property on the first access, rather than calculating and copying them up front. This is a really big win with properties that may force layout

that the event handler may not even need. The most common use we know of was to add properties for pointer events, which is no longer necessary because those events are supported already in jQuery 3.0. The jQuery Migrate plugin provides support for these properties if you still need them. The related but undocumented `mouseHooks` and `keyHooks` lists were removed as well. The team is interested in understanding other use cases before defining new APIs, so feel free to open a ticket.

https://github.com/jquery/jquery/issues/3103
https://github.com/jquery/jquery/issues/1746
https://learn.jquery.com/events/event-extensions/

Breaking change: Delegated events with bad selectors throw *immediately*

Before jQuery 3.0, the selector used in a delegated event wasn't used until the first time that event occurred on the element. This sometimes led to hard-to-debug cases where the error was far removed from the time and code where the mistake was made. Now the selector is tested when the event is attached and throws an error if it is not valid. The only reason this is considered a breaking change is that code might use an invalid selector when attaching an event that never occurs, and that case previously would have never thrown an error.

https://github.com/jquery/jquery/issues/3071

Deprecated: .bind() and .delegate()

Five years ago in jQuery 1.7 we introduced the `.on()` method for attaching event handlers. The older `.bind()`, `.unbind()`, `.delegate()` and `.undelegate()` methods are being deprecated as of 3.0, but are still present. The API documentation explains how to rewrite the calls using the `.on()` and `.off()` methods.

Manipulation

Breaking change: `.wrapAll(function)` only calls the function once

In previous versions, the `.wrapAll()` method acted like `.wrap()` when a function was passed. This has been corrected; now and `.wrapAll(function)` calls its function once, using the string result of the function call to wrap the entire collection.

https://github.com/jquery/jquery/issues/1843

Offset

Breaking change: Invalid input to the `.offset()` method

When using the `.offset()` method, the first item in the jQuery collection must be a DOM element that has a DOM `getBoundingClientRect()` method. (All browsers supported by jQuery 3.0 have this API.) Any other input may result in jQuery throwing an error. Also note that the element must be visible and currently in a document (i.e., not disconnected).

https://github.com/jquery/jquery/issues/2115
https://github.com/jquery/jquery/issues/2114

Selector

Breaking change: Behavior of `:hidden` and `:visible`

An element is considered now visible if it has a layout box returned from the DOM `getClientRects()` method, even if that box has a height and/or width of zero. This means that elements such as `<br />` or an empty `<span>` element that don't have height are considered to be visible.

https://github.com/jquery/jquery/issues/2227
https://github.com/jquery/jquery/issues/2604

Breaking change: `jQuery("#")` and `.find("#")` are invalid syntax

jQuery 3.0 throws a syntax error if a selector string consists of nothing but a hash-mark. In previous versions, `$("#")` returned an empty collection and `.find("#")` threw an error.

https://github.com/jquery/jquery/pull/1682

Feature: New method `jQuery.escapeSelector()`

The new `jQuery.escapeSelector( selector )` method takes a selector string and escapes any character that has a special meaning in a CSS selector. It is essentially a shim for the CSS Working Group's `CSS.escape()` method that runs on all of jQuery's supported browsers. This method is useful for situations where a class name or an ID contains characters that have a special meaning in CSS, such as the dot or the semicolon.

For example, if an element on the page has an id of "abc.def" it cannot be selected with `$( "#abc.def" )` because the selector is parsed as "an element with id 'abc' that also has a class 'def'. However, it *can* be selected with `$( "#" + $.escapeSelector( "abc.def" ) )`.

https://github.com/jquery/jquery/issues/1761

Deprecated: `jQuery.expr[":"]` and `jQuery.expr.filters`

These two names for defining custom selectors through jQuery's Sizzle selection engine are the same as `jQuery.expr.pseudos`, so we are deprecating the redundant names.

Serialize

Breaking change: jQuery.param() no longer converts %20 to a plus sign

In forms that are POST-ed via `jQuery.ajax()`, the specification for `application/x-www-form-urlencoded` encoding says that any occurrence of a space should be converted to the `+` character. Previously, jQuery implemented this by doing the conversion in `jQuery.param()`. Now, that conversion has been moved into `jQuery.ajax()`. Data encoded via `jQuery.param()` will convert spaces to `%20`, which makes it directly compatible with the use of the native JavaScript `encodeURIComponent()` and `decodeURIComponent()` methods.

https://github.com/jquery/jquery/issues/2658

Traversing

Breaking change: `.andSelf()` removed, use `.addBack()`

The `.andSelf()` method was deprecated in jQuery 1.8 and now removed in 3.0 in favor of the `.addBack()` method, which does a better job of explaining what it does and also accepts an optional selector to filter what is added back.