How to create a basic plugin

The notation for creating a typical plugin is as follows:

(function($){

$.fn.myNewPlugin = function() {

return this.each(function(){

// do something

});

};

}(jQuery));

Don't let that confuse you though. The point of a jQuery plugin is to extend jQuery's prototype object, and that's what's happening on this line:

$.fn.myNewPlugin = function() { //...

We wrap this assignment in an immediately-invoked function:

(function($){

//...

}(jQuery));

This has the effect of creating a "private" scope that allows us to extend jQuery using the dollar symbol without having to risk the possibility that the dollar has been over-written by another library.

So our actual plugin, thus far, is this:

$.fn.myNewPlugin = function() {

return this.each(function(){

// do something

});

};

The this keyword within the new plugin refers to the jQuery object on which the plugin is being called.

var somejQueryObject = $('#something');

$.fn.myNewPlugin = function() {

alert(this === somejQueryObject);

};

somejQueryObject.myNewPlugin(); // alerts 'true'

Your typical jQuery object will contain references to any number of DOM elements, and that's why jQuery objects are often referred to as collections.

So, to do something with a collection we need to loop through it, which is most easily achieved using jQuery's each() method:

$.fn.myNewPlugin = function() {

return this.each(function(){

});

};

jQuery's each() method, like most other jQuery methods, returns a jQuery object, thus enabling what we've all come to know and love as 'chaining' ($(...).css().attr()...). We wouldn't want to break this convention so we return the this object. Within this loop you can do whatever you want with each element. Here's an example of a small plugin using some of the techniques we've discussed:

(function($){

$.fn.showLinkLocation = function() {

return this.filter('a').each(function(){

$(this).append(

' (' + $(this).attr('href') + ')'

);

});

};

}(jQuery));

// Usage example:

$('a').showLinkLocation();

This handy plugin goes through all anchors in the collection and appends the href attribute in brackets.

<!-- Before plugin is called: -->

<a href="page.html">Foo</a>

<!-- After plugin is called: -->

<a href="page.html">Foo (page.html)</a>

Our plugin can be optimised though:

(function($){

$.fn.showLinkLocation = function() {

return this.filter('a').append(function(){

return ' (' + this.href + ')';

});

};

}(jQuery));

We're using the append method's capability to accept a callback, and the return value of that callback will determine what is appended to each element in the collection. Notice also that we're not using the attr method to retrieve the href attribute, because the native DOM API gives us easy access with the aptly named href property.

Here's another example of a plugin. This one doesn't require us to loop through every elememt with the each() method. Instead, we're simply going to delegate to other jQuery methods directly:

(function($){

$.fn.fadeInAndAddClass = function(duration, className) {

return this.fadeIn(duration, function(){

$(this).addClass(className);

});

};

}(jQuery));

// Usage example:

$('a').fadeInAndAddClass(400, 'finishedFading');