# Deferred [jQuery.Deferred()]

It is a chainable utility object created by calling the [jQuery.Deferred()](http://api.jquery.com/jQuery.Deferred/) method.

It can register multiple callbacks into callback queues, invoke callback queues, and relay the success or failure state of any synchronous or asynchronous function.

**Code Optimization**

* Append Outside of Loops
* Cache Length During Loops
* Detach Elements to Work with Them
* Don’t Act on Absent Elements
* Optimize Selectors -
* Use Stylesheets for Changing CSS on Many Elements
* Don’t Treat jQuery as a Black Box

# Append Outside of Loops

This is a common problem when appending elements within a loop.

$.each ( myArray, function ( i, item) {

var newListItem = "<li>" + item + "</li>";

$( "#ballers" ).append( newListItem );

});

One common technique is to leverage a document fragment. During each iteration of the loop, you append the element to the fragment rather than the DOM element. After the loop, just append the fragment to the DOM element.

var frag = document.createDocumentFragment();

$.each( myArray, function( i, item ) {

var newListItem = document.createElement( "li" );

var itemText = document.createTextNode( item );

newListItem.appendChild( itemText );

frag.appendChild( newListItem );

});

$( "#ballers" )[ 0 ].appendChild( frag );

Another simple technique is to build up a string during each iteration of the loop. After the loop, just set the HTML of the DOM element to that string.

var myHtml = "";

$.each( myArray, function( i, item ) {

myHtml += "<li>" + item + "</li>";

});

$( "#ballers" ).html( myHtml );

# Cache Length during Loops

In a for loop, don't access the length property of an array every time; cache it beforehand.

var myLength = myArray.length;

for ( var i = 0; i < myLength; i++ ) {

}

# Detach Elements to Work with Them

The DOM is slow; you want to avoid manipulating it as much as possible. jQuery introduced detach() in version 1.4 to help address this issue, allowing you to remove an element from the DOM while you work with it.

var $table = $( "#myTable" );

var $parent = $table.parent();

$table.detach();

// ... add lots and lots of rows to table

$parent.append( $table );

# Don’t Act on Absent Elements

JQuery won't tell you if you're trying to run a whole lot of code on an empty selection – it will proceed as though nothing's wrong. It's up to you to verify that your selection contains some elements.

*// Bad: This runs three functions before it*

*// realizes there's nothing in the selection*

$( "#nosuchthing" ).slideUp();

*// Better:*

**var** $mySelection = $( "#nosuchthing" );

**if** ( $mySelection.length ) {

$mySelection.slideUp();

}

*// Best: Add a doOnce plugin.*

jQuery.fn.doOnce = **function**( func ) {

**this**.length && func.apply( **this** );

**return** **this**;

}

$( "li.cartitems" ).doOnce(**function**() {

*// make it ajax! \o/*

});

# Optimize Selectors

Selector optimization is less important than it used to be, as more browsers implement document.querySelectorAll() and the burden of selection shifts from jQuery to the browser. However, there are still some tips to keep in mind.

## ID-Based Selectors

Beginning your selector with an ID is always best.

$( "#container div.robotarm" ); // Fast:

$( "#container" ).find( "div.robotarm" ); // Super-fast:

The .find() approach is faster because the first selection is handled without going through the Sizzle selector engine – ID-only selections are handled using

document.getElementById(), which is extremely fast because it is native to the browser.

## Specificity

Be specific on the right-hand side of your selector, and less specific on the left.

$( "div.data .gonzalez" ); *// Unoptimized:*

$( ".data td.gonzalez" ); /*// Optimized:*

Use tag.class if possible on your right-most selector, and just tag or just .class on the left.

## Avoid Excessive Specificity

$( ".data table.attendees td.gonzalez" );

*// Better: Drop the middle if possible.*

## $( ".data td.gonzalez" );

## Avoid the Universal Selector

Selections that specify or imply that a match could be found anywhere can be very slow.

$( ".buttons > \*" ); *// Extremely expensive.*

$( ".buttons" ).children(); *// Much better.*

$( ".category :radio" ); *// Implied universal selection.*

$( ".category \*:radio" ); *// Same thing, explicit now.*

$( ".category input:radio" ); *// Much better.*

# Use Stylesheets for Changing CSS on Many Elements

If you're changing the CSS of more than 20 elements using .css(), consider adding a style tag to the page instead for a nearly 60% increase in speed.

// Fine for up to 20 elements, slow after that:

$( "a.swedberg" ).css( "color", "#0769ad" );

// Much faster:

$( "<style type=\"text/css\">a.swedberg { color: #0769ad }</style>")

.appendTo( "head" );

-----------------------------------------------------------------------------

# Avoiding Conflicts with Other Libraries

The jQuery library and virtually all of its plugins are contained within the jQuery namespace.

*By default, jQuery uses* $ *as a shortcut for* jQuery

If you are using another JavaScript library that uses the $ variable, you can run into conflicts with jQuery. In order to avoid these conflicts, you need to put jQuery in no-conflict mode immediately after it is loaded onto the page and before you attempt to use jQuery in your page

1. Create new alias and then use the alias:

var $j = jQuery.noConflict();

1. Use Immediately invoke function:

jQuery.noConflict();

(function( $ ) {

// Your jQuery code here, using the $

})( jQuery );

1. Use $ as an argument passed to jquery ready function :

jQuery(document).ready(function($)

{

// Your jQuery code here, using the $

});

**AJAX (Asynchronous Javscript and XML):**

Because the request is asynchronous, the rest of your code continues to execute while the request is being processed.

JQuery offers both a full-featured $.ajax() method, and simple convenience methods such as $.get(), $.getScript(), $.getJSON(), $.post(), and **$.load().**

**jQuery load()** method is a simple, but powerful AJAX method.

The load() method loads data from a server and puts the returned data, e.g. HTML fragment, into the selected element.

***Syntax:*** *$(selector).load(URL,data,callback);*

e.g. $(“#myDiv”).load(“http://server/test.html”,function(responseTxt, statusTxt, xhr){

//This is call back handler.

});

Ajax does not work across domains. As a work around, use JSONP (JSON with Padding).

Key Concepts:

1. **Type**: Get and Post
2. **Data types**[Type of data accept from Ajax response] :

Text, xml, JSON, JSONP, script, html,

e.g. $.ajax({

url: "post.php", // the URL for the request

data: { id: 123 }, // the data to send

type: "GET", // whether this is a POST or GET request

dataType : "json", // the type of data we expect back

success: function( json ) { // Do something },

error: function( xhr, status ) {

alert( "Sorry, there was a problem!" );

},

complete: function( xhr, status ) { // code to run regardless of success or failure

alert( "The request is complete!" );

}

});

**XHR** – XmlHttpRequest

**JSON** - Javscript Object notification

**Ajax Events:**

**Local Events** are callbacks that you can subscribe to within the Ajax request object

**Global events** are triggered on the document.

**ajaxStart** (Global Event)

This event is triggered if an Ajax request is started and no other Ajax requests are currently running.

**beforeSend** (Local Event)  
This event, which is triggered before an Ajax request is started, allows you to modify the XMLHttpRequest object (setting additional headers, if need be.)

**ajaxSend** (Global Event)  
This global event is also triggered before the request is run.

**success** (Local Event) and **ajaxSuccess** (Global Event)  
This event is only called if the request was successful (no errors from the server, no errors with the data).

**error** (Local Event) and **ajaxError** (Global Event)  
This event is only called if an error occurred with the request (you can never have both an error and a success callback with a request).

**complete** (Local Event) and **ajaxComplete** (Global Event)  
This event is called regardless of if the request was successful, or not. You will always receive a complete callback, even for synchronous requests.

**ajaxStop** (Global Event)  
This global event is triggered if there are no more Ajax requests being processed.

**State of Ajax request:**

*xmlhttp.****onreadystatechange****=function(){  
 if (xmlhttp.****readyState****==4 && xmlhttp.status==200)  
    {  
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;  
    }*

*}*

The **readyState** property holds the status of the XMLHttpRequest.

Status changes from 0 to 4:   
0: request not initialized  (uninitialized)  
1: server connection established (loading)  
2: request received (loaded)  
3: processing request (interactive)  
4: request finished and response is ready (complete)

The jquery ajax method returns a [**XMLHttpRequest**](http://www.w3.org/TR/XMLHttpRequest/) object. You can use this object to cancel the request.

Clone of any object:

 $('#dvText').clone().appendTo('body');

 $('#dvText').clone(true).appendTo('body'); // Copy event as well.