Tutorials:How jQuery Works

http://docs.jquery.com/Tutorials:How_jQuery_Works

Original: http://docs.jquery.com/Tutorials:How_jQuery_Works
Author: John Resig.

Similar Tutorials: jQuery Core, Selectors, CSS, Traversing, Manipulation, Events, Effects

CONTENTS

[hide]

1 jQuery: The Basics
2 Launching Code on Document Ready
3 Complete Example
4 Adding and Removing an HTML Class
5 Special Effects
6 Callback and Functions
6.1 Callback without arguments
6.2 Callback with arguments
- 6.2.1 Wrong
- 6.2.2 Right

7 More Reading

jQuery: The Basics

This is a basic tutorial, designed to help you get started using jQuery. If you don't have a test page set up yet, start by creating a new HTML page with the following contents:

```
<!doctype html>

<html>

  <head>

    <meta charset="utf-8">

    <title>Demo</title>

  </head>

  <body>

    <a href="http://jquery.com/">jQuery</a>

    <script src="jquery.js"></script>

    <script>



    </script>
```

```
  </body>

</html>
```

Edit the `src` attribute in the script tag to point to your copy of jquery.js. For example, if jquery.js is in the same directory as your HTML file, you can use:

```
<script src="jquery.js"></script>
```

You can download your own copy of jQuery from the Downloading jQuery page

Launching Code on Document Ready

The first thing that most Javascript programmers end up doing is adding some code to their program, similar to this:

```
window.onload = function(){ alert("welcome"); }
```

Inside of which is the code that you want to run right when the page is loaded. Problematically, however, the Javascript code isn't run until all images are finished downloading (this includes banner ads). The reason for using window.onload in the first place is that the HTML 'document' isn't finished loading yet, when you first try to run your code.

To circumvent both problems, jQuery has a simple statement that checks the `document` and waits until it's ready to be manipulated, known as the ready event:

```
$(document).ready(function(){

  // Your code here

});
```

Inside the ready event, add a click handler to the link:

```
$(document).ready(function(){

  $("a").click(function(event){

    alert("Thanks for visiting!");

  });

});
```

Save your HTML file and reload the test page in your browser. Clicking the link on the page should make a browser's alert pop-up, before leaving to go to the main jQuery page.

For click and most other events, you can prevent the default behaviour - here, following the link to jquery.com - by calling event.preventDefault() in the event handler:

```
$(document).ready(function(){

  $("a").click(function(event){

    alert("As you can see, the link no longer took you to jquery.com");
```

```
    event.preventDefault();

  });

});
```

Complete Example

The following is an example of what the complete HTML file might look like if you were to use the script in your own file. Note that it links to Google's CDN to load the jQuery core file. Also, while the custom script is included in the <head>, it is generally preferable to place it in a separate file and refer that file with the script element's src attribute

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>jQuery demo</title>

</head>

<body>

  <a href="http://jquery.com/">jQuery</a>

  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.5/jquery.min.js"></script>

  <script>

    $(document).ready(function(){

      $("a").click(function(event){

        alert("As you can see, the link no longer took you to jquery.com");

        event.preventDefault();

      });

    });

  </script>

</body>

</html>
```

Adding and Removing an HTML Class

Important: *The remaining jQuery examples will need to be placed inside the ready event so that they are executed when the document is ready to be worked on. See Launching Code on Document Ready above for details.*

Another common task is adding (or removing) a `class`.

First, add some style information into the <head> of your document, like this:

```
<style>

   a.test { font-weight: bold; }

</style>
```

Next, add the addClass call to your script:

```
 $("a").addClass("test");
```

All your `a` elements will now be bold.

To remove the `class`, use removeClass

```
$("a").removeClass("test");
```

(HTML allows multiple classes to be added to an element.)

Special Effects

In jQuery, a couple of handy effects are provided, to really make your web site stand out. To put this to the test, change the click that you added earlier to this:

```
$("a").click(function(event){

  event.preventDefault();

  $(this).hide("slow");

});
```

Now, if you click any link, it should make itself slowly disappear.

CALLBACK AND FUNCTIONS

A callback is a function that is passed as an argument to another function and is executed after its parent function has completed. The special thing about a callback is that functions that appear after the "parent" can execute before the callback executes. Another important thing to know is how to properly pass the callback. This is where I have often forgotten the proper syntax.

Callback *without* arguments

For a callback with no arguments you pass it like this:

```
$.get('myhtmlpage.html', myCallBack);
```

Note that the second parameter here is simply the function name (but *not* as a string and without parentheses). Functions in Javascript are 'First class citizens' and so can be passed around like variable references and executed at a later time.

Callback *with* arguments

"What do you do if you have arguments that you want to pass?", you might ask yourself.

Wrong

The Wrong Way (will **not** work!)

```
 $.get('myhtmlpage.html', myCallBack('foo', 'bar'));
```

This will not work because it calls

```
myCallBack('foo', 'bar')
```

and then passes the return value as the second parameter to `$.get()`

Right

The problem with the above example is that myCallBack('foo', 'bar') is evaluated before being passed as a function. Javascript and by extension jQuery expects a function pointer in cases like these. I.E. setTimeout function.

In the below usage, an anonymous function is created (just a block of statements) and is registered as the callback function. Note the use of 'function(){'. The anonymous function does exactly one thing: calls myCallBack, with the values of 'foo' and 'bar'.

```
$.get('myhtmlpage.html', function(){

  myCallBack('foo', 'bar');

});
```

myCallBack is invoked when the '$.get' is done getting the page.

MORE READING

From here, you should probably begin looking at the rest of the Documentation - it's very comprehensive and covers all aspects of jQuery. If you have any questions, please feel free to send a message to the jQuery Forums