8. The Big Picture

Contents

[1. Introduction 1](#_Toc3488)

[2. Different jQuery Versions 1](#_Toc29273)

[3. jQuery in Context of Other Libraries 2](#_Toc5957)

[4. Demo: JavaScript Function Scope 3](#_Toc21931)

[5. Demo: Naming Conventions 4](#_Toc24488)

[6. Summary 5](#_Toc19480)

# 1. Introduction

=>slides: Pg. 1

This final module is meant to leave you with the context and perspective you need to continue your learning in jQuery development. Here I’ll share with you some coding tips and explain how jQuery fits into the overall web development ecosystem. Let’s begin by discussing the different versions of jQuery available.

# Different jQuery Versions

=>slides: Pg. 2

jQuery is available in three different versions. What’s the difference between the versions? Well to understand the answer to this question we need to first consider another question…

=>slides: Pg. 3

What you need to know first is, will the users of your web application require support for Internet Explorer 6, 7 or 8? If the answer is yes to any of these versions then you want to select jQuery 3.x Compat.

=>slides: Pg. 4

The API is identical between jQuery 3.x Compat and 3.x Standard, but the only difference is the underlying support for older versions of internet explorer. If your users are not accessing your app with anything but Internet Explorer version 9 or higher

=>slides: Pg. 5

Then you can safely use jQuery 3.x. The fact is, much of the code base of version 3.x compat is busy dealing with shortcomings of older versions of Internet Explorer trying to make it compatible. So by removing all the logic required to make IE behave, the file size of jQuery is drastically reduced and is recommended when you're building modern web applications. Now beyond different versions, you'll often find jQuery in a number of different contexts. So next, we'll examine jQuery's relationship to other libraries.

# jQuery in Context of Other Libraries

=>slides: Pg. 6

In order to provide context for what you are learning about jQuery, I want to have a clear understanding of how jQuery lives in the larger JavaScript ecosystem. [click] Therefore during the next few sets of slides I want to paint a picture of how and where you will encounter JavaScript and how jQuery fits into this scenario.

=>slides: Pg. 7

jQuery resides in a rich JavaScript ecosystem. On the client, jQuery is often the foundation for scores of web applications and is the basis for component development found in jQuery UI. jQuery UI…

=>slides: Pg. 8

jQuery UI is a collection of user interface interactions, effects, widgets, and themes all built on top of the jQuery. Some of the controls (or “widgets” as known in jQuery UI) include: datepicker, menu, tabs, a progress bar and more.

=>slides: Pg. 9

This sets the foundation for further development for custom widgets, plug-ins and application code that you may write or might be available via open source and commercial libraries. The bottom line is that jQuery itself is so versatile that you can almost anything you need in a web application from the simple to full-featured commercial products.

=>slides: Pg. 10

Now, what happens when you take the same JavaScript engine that runs in the web browser [click] and allow it to run natively on desktops and servers? That is exactly what node.js is. Node.js allows you to write JavaScript for desktop and server applications without the need of a web browser. Using Node you have access to the computers resources like file system, etc. What this means is that while you can use jQuery in web applications you can also use jQuery in HTML-based native applications as well.

=>slides: Pg. 11

The last piece of the puzzle is writing mobile and device-specific applications. There is a library you should know about called Zepto. [click] Straight off the website: Zepto is a minimalist JavaScript library for modern browsers with a largely jQuery-compatible API. If you use jQuery, you already know how to use Zepto.

So while the distinction between jQuery 1 and 2 is a departure from supporting old desktop browsers (IE 8 and below), Zepto is a much smaller file in terms of bytes and is optimized for mobile devices. What’s important for you to know right now about this is – again – if you know jQuery then you know zepto – Zepto is just most appropriate for mobile web applications.

Now – before you dig in to writing your first jQuery application I’d like to review some coding techniques that will prove helpful as you continue your development.

# Demo: JavaScript Function Scope

=>slides: Pg. 12

Before I wrap up this course, there are a few coding tips that I would like to share with you that I think will make your life a little bit easier, and hopefully help you avoid some common bugs. Now this tip is dealing with function scope, and this really is a subject that's much broader than jQuery itself, but since we spend so much time creating anonymous functions within jQuery, I want to refresh your memory, if this is something you've already learned, or introduce you to it, if it's something new to you, about how function scope works in JavaScript.

=>slides: Pg. 13

Now if we take a look at this variable here, name = Craig, and then I have a function immediately after that, where logName, when I call logName, it'll log my name to the console. So let's go ahead and call this function, and when we run it on the page, you'll see that what it does is it logs my name to the console window. Now this works like this, even though the name variable is not declared anywhere within the logName function because it's all running under the same context. And right now, these are both running in the global scope, and global scope just means that the name variable and the logName function are kind of available anywhere within the running program. So let's take a look at another example as well.

=>slides: Pg. 14

Here I've pasted in a new function, and this one logs out my Twitter handle. So if I call logTwitter, and run this on the page, you'll notice that that logs out my Twitter handle. But if we look carefully at this function, you'll notice that the variable for my Twitter handle is declared inside of the logTwitter function. So what would happen if I try to, outside of the scope of the logTwitter function, try to report back the value of the Twitter variable? Well, let's try it and see. So now I'll refresh the page, and you'll notice that I get a reference error. It says that the Twitter variable is not defined. It's because the Twitter variable is defined inside the logTwitter function, and since it's declared inside the function, once that function is done, and it's out of scope, that variable goes away. So you only have access to variables that are declared within a scope that's current.

=>slides: Pg. 15

So let's try one more just to drive home the point a little more clearly. So here I've pasted in another function for us to look at. So this is called logFullName. So I'm starting off with a variable called first, which has my first name, and then inside of logFullName is another function declaration. So this is called logName. Inside of there, I have a new variable called last and that has my last name, and then I'm doing console.log, and concatenating together my first name and my last name. So that defines the function, and so then I just run it at the end of this function. So now let's try running logFullName and see what happens. So you'll notice that the result prints out my full name to the console, but notice what's happening here. The first name variable is declared inside the logFullName function, but then it's used inside of the logName function, and that's perfectly acceptable and perfectly expected because it's within scope of the full logName function. So, basically, if you have a variable that's in scope within the function that's running, it will be available to any inner functions that are also running in that context.

=>slides: Pg. 16

For more in‑depth study on this subject, I suggest you check out Ryan Morr's article, Understanding Scope and Context in JavaScript. Now, there's one more thing I want to leave you with, and that's a naming convention that can help you write more accurate jQuery code.

# Demo: Naming Conventions

=>slides: Pg. 17

The next tip that I'd like to share with you is a variable naming convention that can help you sort out your jQuery code pretty easily, just as you're looking at it. So first, let's take a look at the code that we would write to select an element on the page using just regular everyday JavaScript.



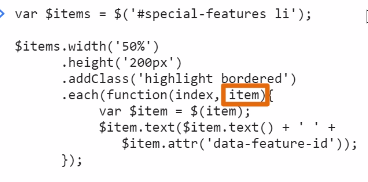
So here I've used getElementsByTagName in order to select the h2 on the page. And so what's returned here is the h2, that's found right there, the address that we've been using all throughout the course. Now you'll notice that the name of the variable that I've called it here is just simply h2.



Now, if I wanted to make that same type of selection using jQuery, normally, when I'm writing regular production code, I would name that a little bit different, and what I'll do is prefix that with a dollar sign. So this is the selection that I would use in the variable name, so $h2 would be equal to the jQuery selector of h2.



And so when we take a look at the results, of course we get the same element back, but this is a jQuery wrapped set. Now the value for this is that so I can tell by looking at my variable which one is the result of a jQuery selection and which one is the result of standard JavaScript selection. Now, why would you need this? Why would you even care? Well, because in many different circumstances, sometimes you'll be dealing with raw DOM elements, and Sometimes you'll be working with jQuery wrapped sets, and even the same elements at the same time, and so by using the dollar sign prefix, it can help make your code a lot easier to understand, and, in fact, earlier in the course, you saw an example of where that became valuable.



If you recall, the chaining example featured code that looked a lot like this. Now you'll notice during the each function, the iterator for the each function, one of the arguments is the specific list item that's being returned as the each function is looping through those items. Well, the item in the argument is the raw list item, the raw DOM element item, and so here I've passed in that item, then the jQuery selection, and now the variable is $item, and even at the top, you'll see that the collection itself of all of the items I prefixed with $items. And so you'll find different circumstances in which sometimes you're dealing with the raw DOM element and sometimes you're working with the jQuery selected elements, and you'll want to be able to differentiate with them, and by using the prefix of the dollar sign, it can make your life a whole lot easier.

# Summary

=>slides: Pg. 18

In this module I set jQuery in context of the JavaScript ecosystem as a whole

=>slides: Pg. 19

and shown you how where it sits in relation to a number of other JavaScript libraries

=>slides: Pg. 20

Beyond that I shared a few coding techniques that will help you avoid errors and set you up for forming good coding habits as your development skills become more sophisticated.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*