JavaScript & HTML

Modifying HTML

Since JavaScript was created for use in the browser, it is very commonly integrated into HTML. To allow JavaScript code to be correctly recognised in HTML, the <script> tag must be used before and after JavaScript code.

The HTML <script> tag by default specifiys JavaScript, since HTML5 the explict type of code is not required in the tag. JavaScipt code is then placed between the tags and closed off with </script>

Scripts can be added inline if short, however generally for performance they are external being referenced by URL using the ‘src’ attribute. The ‘src’ attribute will search specific locations for the script depending on the environment setup, but the public folder is generally where JavaScript files are kept:

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

Dynamic Script Tag Injection can be used to create new <script> elements during runtime, creating dynamic pages. When a new script tag is created its ‘src’ URL is automatically downloaded and executed. New script elements are created using ‘document.createElement()’ functions, then setting the attributes to ‘src’ and relevant URL for the new script. Finally the new script element is added to the document head, causing it to be downloaded and executed.

*ar script = document.createElement("script");*

*script.setAttribute("src", <url>);*

*document.head.appendChild(script);*

Async in Browsers

When the browser reads the html and comes across a <script> tag, it starts downloading an executing the script. This is called synchronous blocking, and during this period of downloading and executing, the rest of the page loading is blocked, making the page seem unresponsive to the user if the script is large. To mitigate this use the “async=true” attribute for the <script> tag, which should unblock the rest of the page to load whilst the javascript is running.

<script src="file.js" async="async"></script>

Async should only be used with external JavaScript files. Also if scripts are dependent on other scripts, ie one downloads a library, then another calls from it, both should have async set to false to avoid errors in timing if the library downloads slowly.

For JavaScript files which need to be run once the DOM has been set up fully, use the event listener for load:

*window.addEventListener("load", function() {*

*//add function*

*}*

Changing HTML Content

One of the most commonly used functions of javascript is changing HTML automatically without having to refresh the page. To do this HTML elements are selected by id then reassigned a new value:

*document.getElementById("demo").innerHTML = "Hello JavaScript"*

*document.getElementById("demo").style.fontSize = "35px"*

*document.getElementById("demo").style.display = "none"*

*document.getElementById("demo").style.display = "block"*

General Methods

event.preventDefault() - Overrides and stops default action happening on an element, such as stopping a checkbox from checking, or to create a warning when specific charaters are typed into a textbox.

Single Page Frontend Apps

The essence of a frontend single page app, is that once the user loads up the page, it never refreshes. To do this single page apps use JavaScript event listeners to dynamically create and add HTML nodes depending on the state and event which occurs.

Changing the URL (window location) is performed using hashes (#), meaning the page wont refresh, however the content will change. Content change is done by evaluating the ‘window.location’ ‘.hash’ value. and making changes depending on that string. Using <a href=’#<value>’> allows links to change the hash value of the window, and in turn dynamically changing the content.

Model View Controller - Frontend apps

Unlike in backend apps, JavaScript frontend webapps don’t really have convetions to how concerns are seperated and handled. Therefore, there are various different ways of handling events and the state of the site.

Generally the site will start with a very little in the DOM (i.e. one div/section) which the JavaScript included on the page then fills with the relevant nodes, setting up events for user interaction. From there concerns are separated in various different ways:

Traditional MVC:

* Controller: takes user interaction, modifying the models sending the new state to the view to update the page DOM
* View: Generates node and updates the page DOM with information given by the controller
* Model: Handles webapp state

Model & View Combined:

* Controller: handles user input telling the model/view to create and return new HMTL nodes based on input, controller then inserts nodes into required location. Since the controller knows the location of user interaction, this method means that the location of elements doesn’t have to be passed between controller and view
* Model/View: Handles state and generation of the new DOM nodes which will be updated. This method means that return values can easily be tested, and state is not unessecarily passed from model to controller to view, since it is all encapsulated in the model/view.

XML Requests

Whereas standard HTTP request are synchronous and wait for a response then make a page reload with that reponse, JavaScript has the ability to make XMLHttpRequests, which can be used asyncronhously without page reloads. This allows for JavaScript to make requests for XML and JSON data in the background and use the response to dynamically update the page without refreshes.

XML requests are generally used for returning JSON or XML for many reasons:

* JSON/XML is more compact, meaning better performance for requests
* Some browser have bugs with certain data, returning only JSON/XML means bugs can be handled in one place instead of where-ever the other data type reponse is placed
* Conformitiy in request means that the server can have a single implementation returning JSON no matter if the user is a function (API call) or human
* Easier testing when the server doesn’t have to generate HTML
* Easier to style the HTML if it is generated locally

to add:

* since a function is also a variable if a function is defined inside another function, it will only have the scope of the function it is defined in. If the function is then returned from its scope function it is called a closure and reduces excessive functions being available in the gobal scope
* event listeners
* set timeout - waits until after function completed, race condition several things run but not knowing what comesback first
* set interval
* what is jquery