Introduction to jQuery

Relevant Links

- Flanagan's book, Chapter 19
- Flanagan's book, Part IV (jQuery summary around page 945)
- devdocs on jQuery[1]
- jQuery front page[2]
- jQuery API[3]

Notes

jQuery is a very popular library for handling the DOM parts of an application. It offers a uniform interface, not marred by the various incompatibilities between browser implementations.

jQuery has 3 main components that we will utilize in due course:

- Page / element access and manipulation
- event-handling
- asynchronous data loading

Even if you do decide not to use it, you need to become familiar with it, as many other libraries offer a similar interface.

As an external library, jQuery needs to be first incorporated into your project via a script tag (or as a module as will learn later). You can either download the version you want and link to it, or you can link to the jQuery CDN[4] (the wikipedia page[5] is a good place to start if you want to learn more about Content Delivery Networks and their advantages). So the simplest way to get jQuery into your code is to include a script tag like the following

```
<script
  src="https://code.jquery.com/jquery-3.3.1.min.js"
  integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="
  crossorigin="anonymous"></script>
```

You can also load jQuery on a web page that doesn't have it loaded yet by doing something like the following on the console:

```
{
  let script = document.createElement('script');
  script.setAttribute('src', 'https://code.jquery.com/jquery-3.3.1.min.js');
  document.body.appendChild(script);
}
```

[1] http://devdocs.io/jquery/
[2] http://jquery.com/
[3] http://api.jquery.com/
[4] https://code.jquery.com/
[5] http://en.wikipedia.org/wiki/Content_delivery_network

You can follow these notes along by using the sample found here[6].

jQuery defines two global objects, that you can use to access it. One is jQuery, the other is $. For all our purposes, they are synonyms and we will be using $.

The global (factory) function $

The global $ is a swiss-army knife, it is a function that sort of does everything. It has three main forms, depending on what the first argument is:

$(sel) If the first argument is a string that is a selector, it will return a "jQuery object" that represents the collection of all elements that match the selector. If a second "context" argument is passed, representing an element or elements, it will search for the elements *relative to that context*.

For example in our sample page above, $("h2", "#sidebar") will target the h2 elements that are within the element with id #sidebar.

$(html) If the first argument is a string that is html text, then it will create and return a new object representing that html code. This object is currently detached (does not show up on the web page), but it can then be inserted into some point in the web page specified by future commands.

For example the following creates a new list item with a link in it:

let newLink = $('
Google!
'); We can then add it to the list in the sidebar by doing: $('#sidebar ul').append(newLink);

$(domEls) If the first argument is a DOM element or array of DOM elements, it will return a "jQuery object" that represents that same array of elements, but now as jQuery objects.

$(f) If the first argument is a function, then this function will be called when the document has finished loading. This is a good place to do page initialization.

NOTE: This is an important note. Many elements on the page are not accessible via Javascript until the page has been fully loaded. This does not happen until after all the script tags in the file have been fully executed. Therefore scripts that want to interact with the webpage must set up a callback function to be executed *after* the page has been fully loaded. jQuery makes this easy.

[6] ../samplePage/test.html

Examples

Let's assume that jQuery was loaded on the sample web page we were looking at last time. We start by locating the `ul` element on the sidebar that contains our navigation links:

```
var linkList = $('#sidebar ul');
```

We will now create a new element and append it to that list:

```
$('<li><a href="http://www.google.com">Google!</a></li>')
  .appendTo(linkList);
```

Next we pick out the second `li` from within the list, and move it to the bottom of the list:

```
var el2 = $('li', linkList)[1];
linkList.append(el2);
```

We will learn about these methods in the rest of this section.

jQuery object methods

All jQuery objects respond to a similar set of methods. You can explore the whole API online[7], and in Part IV of Flanagan's book.

In general all jQuery methods operate on an "array/collection of jQuery objects".

Collection methods First we have collection methods.

each(f)[8] Given a function `f(i, el)`, will call the function for each element (if a collection). Note that the arguments on this function are unlike those for `Array.forEach` in standard Javascript; they are reversed.

get(i), el[i][9] gets the element at the i-th index. jQuery objects tend to be collections, this picks out a specific element from that collection. The resulting object is not a jQuery object. If you want to carry out jQuery methods on it, you must wrap it in `$()`.

map(f)[10] Translates all items of the array to a new array, via the function.

filter()[11] Filters the list based on the first argument. See the documentation for options.

add()[12] Adds more elements to the current selection, and returns the resulting object.

children()[13] Selects the children of each node in the current selection.

closest()[14] Returns the closest ancestor of each element that matches certain criteria.

find()[15] Selects all descendants that match certain criteria.

[7] http://api.jquery.com/

Element editing Next we have some element methods that read/write element properties.

addClass[16]**, removeClass**[17]**, toggleClass**[18] Add / remove / toggle classes to each element in the collection.

This can actually toggle multiple classes at once.

attr[19] Get or set one more more attributes (the key-value pairs that appear within the opening HTML tags, like href).

data[20] Get or set data attributes. Data attributes do not affect the visual appearance of an element, but they are a convenient way to attach some Javascript information to specific DOM elements.

val[21] Get or set the "value" associated with the element. Most useful for input elements.

Tree manipulation Next we have methods for inserting/removing elements from the dom.

after()[22]**, before()**[23] Insert the provided content after / before each element.
append()[24]**, prepend()**[25] Appends / prepends the provided content at the end of each element.
appendTo()[26]**, prependTo()**[27] Inserts each element at the end / start of the argument.
html()[28] Sets/gets the html contents of the elements.
insertAfter()[29]**, insertBefore()**[30] Inserts each element after the argument. It will also clone them if the target argument is many elements.
wrap[31]**, wrapInner**[32] Various ways or wrapping specific HTML around/inside the elements.

There are a lot of other methods, mostly related to events, that we will revisit later.

Examples

We will run some examples on the sample page we created already. Remember to add jQuery to it as we described earlier.

Okay, now first off, let's type `$` in the console. We should see that it exists and is a function.

Try each of these lines in order.

```
let mainContent = $("#content");
mainContent.addClass("foo");                    // Add a class to mainContent
let art = $('article', mainContent).first();    // The first article
art.text();                                     // Text in art
art.html();                                     // HTML in art
$('<p>A new first paragraph!</p>')              // Create a new paragraph
```

```
  .hide()                                   // Start it as hidden
  .insertBefore($('p', art).first())        // Insert before 1st article's 1st par
  .show(2000);                              // Slowly make it appear
let items = [
  { name: "chair", quantity: 4, price: 5 },
  { name: "table", quantity: 1, price: 15 },
  { name: "lamp", quantity: 1, price: 15 }
];
let rows = items.map(function(item) {
  let entries = [item.name, item.quantity, item.price]
    .map(function(txt) { return `<td>${txt}</td>`; })
    .join('');
  return `<tr>${entries}</tr>`;
});

let tbody = $('<table><thead><tr><th>Name</th><th>Quantity</th><th>Price</th></tr></thead><tb
  .appendTo($('#content'))
  .find('tbody');
$(tbody).append(rows);

// Now we find each td in the table, and replace the contents with an input field:
$('td').html(function() {
  // "this" here refers to the element itself.
  console.log($(this).html());
  return '<input type="input" value="' + $(this).html() + '" />';
});

// Prints out all the values from all inputs
$('input').each(function() { console.log($(this).val()); });
// Arrow functions can't use "this".
// Most jquery methods that  expect functions provide those functions
// with the index in the list and the element.
$('input').each( (index, element) => console.log($(element).val()) );

$('article').last()     // Find last article
  .find('p').last()     // Grab its last paragraph
  .css('background-color', 'red')        // Set its background color
  .hide(2000)
  .show(2000);
```

Let's look at a longer example. It adds 10 randomly generated numbers that fade out and back in.

```
{
  let ul = $("<ul></ul>").appendTo("#content");
  for (let i = 0; i < 10; i += 1) {
    let v = Math.random();
    $("<li>" + v + "</li>")
      .addClass(v > 0.5 ? "big" : "small")
      .appendTo(ul);
  }
  $('li').each(function(i, v) {
    $(v).fadeOut(Math.random() * 5000, function() {
      $(v).fadeIn(Math.random() * 5000);
    });
  });
}
```

And another. This one adds a new div with "bar" divs in it, each of them with random lengths. This looks like a histogram. It then randomly changes those lengths in an animated process.

```
{
  let parent = $('<div id="bars"></div>').appendTo("#content");
  for (let i = 0; i < 10; i += 1) {
    let v = Math.random();
    $("<div></div>")
      .css({
        'background-color': 'blue',
        height: '10px',
        width: (Math.random() * 300) + 'px'
      }).appendTo(parent);
  }
  $('#bars div').each(function() {
    $(this).animate({ width: (Math.random() * 300) + 'px' });
  });
}
```

Practice problems

1. Find the second paragraph of the first article and set its color to red.
2. Find all h2 elements, and insert right after each an `h3` element with the exact same text.
3. Find all h2 elements, and add an explamation mark to the end of the text.
4. Grab the text out of all paragraph elements, form an array of the texts.
5. Find all paragraph texts, and if any of them have text longer than 50 characters: Hide the paragraph, and insert above it a paragraph with class "condensed" and with content those first 50 characters, followed by three dots (ellipsis).
6. Find the first paragraph of the first article. Animate its top padding from its current value to 5ems, then back.