Interactive Web Development

DOM Manipulation

Dr Russ Ross

Dixie State University—Computer and Information Technologies

Spring 2015
Reading: *Learning jQuery 1.3*, Chapter 5
Manipulating attributes

We manipulate the list of CSS classes associated with a list of elements using three methods:

- `.addClass()`: ensure that the elements have a certain class
- `.removeClass()`: ensure that the elements do not have a certain class
- `.toggleClass()`: alternating between having and not having a certain class

The list of classes is just a space-separated list of names in the `class` attribute of an HTML element. We can set and remove arbitrary attributes as well:

- `.attr()`: set the values of one or more attributes
- `.removeAttr()`: delete one or more attributes
<h1 id="f-title">Flatland: A Romance of Many Dimensions</h1>
<div id="f-author">by Edwin A. Abbot</div>
<h2>Part 1, Section 3</h2>
<h3 id="f-subtitle">Concerning the Inhabitants of Flatland</h3>
<div id="excerpt">an excerpt</div>
<div class="chapter">
<p class="square">Our Professional Men and Gentlemen are Squares (to which class I myself belong) and Five-Sided Figures or <a href="http://en.wikipedia.org/wiki/Pentagon">Pentagons</a>.</p>
<p class="nobility hexagon">Next above these come the Nobility, of whom there are several degrees, beginning at Six-Sided Figures, or <a href="http://en.wikipedia.org/wiki/Hexagon">Hexagons</a>, ...</p>
<p class="pull-quote">It is a Law of Nature with us that a male child shall have <strong>one more side</strong> than his father, so that ...</p>
...<p>}</p>
</div>
Non-class attributes

To set the same attribute to the same value in many elements, we use implicit iteration as usual:

```javascript
$('div.chapter a').attr({ rel: 'external' });
```

As with `.css()`, we can modify multiple attributes at the same time by setting multiple properties in a map. Some attributes may need to be custom to each element, for example the id:

```javascript
$('div.chapter a').each(function (index) {
    $(this).attr({
        rel: 'external',
        id: 'wikilink-' + index
    });
});
```

`.each()` acts as a more convenient for loop, running the given function once for each element.
Iteration using `each()`

The `title` attribute acts as a popup tooltip. We can add one to each link that refers to Wikipedia:

```javascript
$('div.chapter a[href*=wikipedia]').each(function (index) {
    var $thisLink = $(this);
    $thisLink.attr({
        rel: 'external',
        id: 'wikilink-' + index,
        title: 'learn more about ' + $thisLink.text() + ' at Wikipedia'
    });
});
```

`each()` supplies a 0-based index parameter to its callback function, numbering the elements that it is iterating over.

`this` is set to refer to each element in turn.
The $( )$ factory function revisited

$( )$ can be used to select elements on the page, wrap DOM elements to become jQuery objects, and register \texttt{ready( )} handlers for the page.

In addition, it can be used to create new DOM elements from strings containing HTML. We can add an anchor to the top of the page, and links after each paragraph that take the user back to the top. First the links:

```html
$( '<a href="#top">back to top</a>' )
 .insertAfter( 'div.chapter p' );
```

The first part creates a new DOM element, but does not position it anywhere on the page. The second part places it after each of the selected elements.
.insertAfter() takes an element first, and a selector expression second. We can reverse this using .after():

```javascript
$('div.chapter p').after('<a href="#top">back to top</a>');
```

The key difference comes when chaining additional methods in the same statement:

- .insertAfter() returns the newly created `<a>` element, and chained methods will act on it.
- .after() returns the elements selected by `$('div.chapter p')`, and chained methods will act on them.
Inserting new elements

We can add the anchor as the first element inside the `<body>`:

```
$('<a id="top" name="top"></a>').prependTo('body');
```

“Back to top” links are unnecessary near the top of the document. We can modify our selector to start inserting after the third paragraph:

```
$('<a href="#top">back to top</a>')
  .insertAfter('div.chapter p:gt(2)');
```
Element insertion methods

Methods that act on content:

- `appendTo(selector)`: insert as last element inside $(selector)
- `prependTo(selector)`: insert as first element inside $(selector)
- `insertAfter(selector)`: insert right after $(selector)
- `insertBefore(selector)`: insert right before $(selector)

Methods that act on selectors:

- `append(content)`: insert $(content) as last contained element
- `prepend(content)`: insert $(content) as first contained element
- `after(content)`: insert $(content) immediately after
- `before(content)`: insert $(content) immediately before
Moving elements

In addition to creating new elements, we can manipulate the placement of existing elements on the page.

The text has several footnotes, marked with `<span class="footnote">` tags. Through CSS, they are displayed in italics. To move them to the bottom of the page, between `<div class="chapter">` and `<div id="footer">`:

```javascript
$('span.footnote').insertBefore('#footer');
```

Implicit iteration happens in the order the elements appeared in the DOM. We might be tempted to write:

```javascript
$('span.footnote').insertAfter('div.chapter');
```

but this would result in the footnotes appearing in reverse order.
Displaying footnotes

This moves all of the footnotes to the bottom of the page, but they are all displayed one after another as inline text, rather than displaying each on its own line. We could fix this with CSS rules that display footnote `<span>` tags outside the chapter `<div>`:

```css
span.footnote {
    font-style: italic;
    font-family: "Times New Roman", Times, serif;
    display: block;
}
.chapter span.footnote {
    display: inline;
}
```

But a better way would be to place them inside an ordered list, which will number them and format them for us.
Marking, numbering, and linking the context

First, we mark and number the original locations:

```javascript
$('span .footnote').each(function (index) {
  $(this).before(
    '<a href="#foot-note-' + (index+1) + '" id="context-' + (index+1) + '" class="context">\(^' + (index+1) + '\)</a>');</
});
```

Then we create a list after the chapter:

```javascript
$('<ol id="notes"></ol>').insertAfter('div .chapter');
```
Appending footnotes

Finally we move the footnotes down into the list, complete with a link back to the context. The complete code so far:

```javascript
$('ol id="notes">').insertAfter('div .chapter >');
$('span .footnote >').each(function (index) {
    $(this)
        .before(
            '<a href="#foot-note-" + (index+1) + 
            " id="context-" + (index+1) + 
            " class="context"><sup>' + (index+1) + ' </sup > </a>
        .appendTo('#notes ')
        .append('&nbsp;(<a href="#context-" + (index+1) + '" >' + 
            'context </a>)');
});
```

Note that `appendTo()` places the items in the correct order. It is applied to the footnote elements through chaining.

Methods chained after `appendTo()` still apply to the footnote elements, but in their new locations.
Wrapping elements

The footnotes are in the correct place, but they are not numbered. They need to be individually wrapped in `<li>` tags.

`wrap(html)` wraps the currently selected elements in the tag supplied as its parameter:

```javascript
$('ol[id="notes"]').insertAfter('div.chapter');
$('span.footnote').each(function (index) {
    $(this)
        .before(
            '<a href="#foot-note-" + (index+1) + 
            '" id="context-" + (index+1) + 
            '" class="context"><sup>' + (index+1) + '</sup></a>)
        .appendTo('#notes')
        .append('&nbsp;' + (<a href="#context-" + (index+1) + '">context</a>))
        .wrap('<li id="foot-note-"'></li>');
});```
Wrapping elements

There are three wrapping methods:

- **wrap(tag)**: wrap tag around each matched element individually, e.g., a `<li>` around each element being put into a list.

- **wrapAll(tag)**: wrap tag around the entire set of matched elements, e.g., a `<div>` around a group of matched `<p>` tags.

- **wrapInner(tag)**: wrap tag around the contents of each matched element individually, e.g., a `<b>` tag to display the contents of each matched `<p>` in bold.
Copying elements

To copy an existing element, use the `.clone()` method:

```
$('div.chapter p:eq(0)').clone();
```

This creates a copy of the element, but does not place it anywhere on the page (similar to element creation). To place it, use the same methods as before:

```
$('div.chapter p:eq(0)').clone().insertBefore('div.chapter');
```

Note that CSS rules that applied in the original location may not apply in the new location.

Clone takes an optional boolean argument. If true, the events bound to the element are also cloned (by default they are not):

```
$('div.chapter p:eq(0)').clone(true).insertBefore('div.chapter');
```
Pull quotes

Our text has snippets marked as pull quotes:

```html
<p><span class="pull-quote">It is a Law of Nature with us</span> that a male child shall have <strong>one more side</strong> than his father</p>, so that ...
```

We would like to copy these and highlight them in the margin as is often done in print publications to catch the reader’s eye. It is marked with a `<span class="pull-quote">`, which is displayed like normal text in the original location.
The style the pull-quote boxes, we add the `.pulled` class to each cloned snippet:

```css
.pulled {
    background: #e5e5e5;
    position: absolute;
    width: 145px;
    top: -20px;
    right: -180px;
    padding: 12px 5px 12px 10px;
    font: italic 1.4em "Times New Roman", Times, serif;
}
```

This positions it 20px above and to the right of the parent element. To make it work, the parent element must have `absolute` or `relative` positioning.
Cloning for pull quotes

Now we can select all the pull quotes and iterate over each one using `.each()`. We set the parent elements to have relative positioning:

```javascript
$('span.pull-quote').each(function (index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
});
```

Next, we clone each `<span>`, add the `pulled` class to the copy, and place it at the beginning of the paragraph (the CSS will reposition it, so it does not matter where in the paragraph it is located):

```javascript
$('span.pull-quote').each(function (index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
    $(this).clone()
        .addClass('pulled')
        .prependTo($parentParagraph);
});
```
Cloning for pull quotes

We would like to drop some words and replace them with elipses (...). These words are marked in the original with `<span class="drop">`.

We should also strip formatting and links from the pull quotes:

```javascript
$('span.pull-quote').each(function (index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
    var $clonedCopy = $(this).clone();
    $clonedCopy
        .addClass('pulled')
        .find('span.drop')
        .html('&hellip;')
        .end()
        .prependTo($parentParagraph);
    var clonedText = $clonedCopy.text();
    $clonedCopy.text(clonedText);
});
```

Note the use of `.find()` .... `.end()` and `.text()` vs. `.html()`.
Rounded corners

For a rounded-corner effect using images, we enclose it in a second `<div>` to accommodate a second background image:

```javascript
$clonedCopy
  // ...
  .prependTo($parentParagraph)
  .wrap('<div class="pulled-wraper"></div>');
var clonedText = $clonedCopy.text(); // ...
```

With the following CSS:

```css
.pulled-wraper {
  background: url(pq-top.jpg) no-repeat left top;
  position: absolute;
  /* ... */
}
pulled {
  background: url(pq-bottom.jpg) no-repeat left bottom;
  position: relative;
  /* ... */
}
```
In a nutshell

1. To **create** new elements from HTML, use the $() factory function.

2. To **insert** new element(s) **inside** every matched element:
   - `.append()`
   - `.appendTo()`
   - `.prepend()`
   - `.prependTo()`

3. To **insert** new element(s) **adjacent** to every matched element:
   - `.after()`
   - `.insertAfter()`
   - `.before()`
   - `.insertBefore()`
In a nutshell

4. To **insert** new element(s) **around** every matched element:
   - `.wrap()`
   - `.wrapAll()`
   - `.wrapInner()`

5. To **replace** every matched element with new element(s) or text:
   - `.html()`
   - `.text()`
   - `.replaceAll()`
   - `.replaceWith()`
In a nutshell

6. To **remove** element(s) **inside** every matched element:
   - `.empty()`

7. To **remove** every matched element and descendents from the document without actually deleting them:
   - `.remove()`