Interactive Web Development

Selectors

Dr Russ Ross

Dixie State University—Computer and Information Technologies

Spring 2015
Selectors

Reading: *Learning jQuery 1.3*, Chapter 2
The Document Object Model is a tree structure. HTML is used to create elements on a page and to describe the relationships between them. We use family-tree terms when discussing trees.

```html
<html>
  <head>
    <title>the title</title>
  </head>
  <body>
    <div>
      <p>This is a paragraph.</p>
      <p>This is another paragraph.</p>
      <p>This is yet another paragraph.</p>
    </div>
  </body>
</html>
```

Parents, children, siblings, descendants, ancestors, etc.
jQuery makes it easier to select elements from the DOM. Almost anything that can be used in a stylesheet to identify elements can be used to select them using jQuery, by wrapping the selector inside a call to the $( ) factory function:

\`
$(\'div#maincolumn p\')
\`

Important note: jQuery always returns a jQuery object, not a native DOM object. You can get to the native object if you need to, but you should always be aware of what kind of object you are working with.

The $ is used by many JavaScript libraries. It is just a convenient alias for jQuery, so you can always write jQuery(\'\ldots\') instead of $(\'\ldots\') to avoid conflicts. There are other tricks we can use to play nicely with other libraries.
Selector building blocks

The most common building blocks of selectors are tag names, IDs, and class names:

<table>
<thead>
<tr>
<th>Selector</th>
<th>CSS</th>
<th>jQuery</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag name</td>
<td>p</td>
<td>$('p')</td>
<td>Selects all paragraphs</td>
</tr>
<tr>
<td>ID</td>
<td>#some-id</td>
<td>$('#some-id')</td>
<td>Selects element with given ID</td>
</tr>
<tr>
<td>Class</td>
<td>.some-class</td>
<td>$('some-class')</td>
<td>Selects everything with given class</td>
</tr>
</tbody>
</table>

Each jQuery selector returns an object representing all matching elements, which may be zero or more. Through **implicit iteration**, it is easy to manipulate a group of elements.
Some unformatted HTML to work with:

```html
<ul id="selected-plays">
    <li>Comedies
        <ul>
            <li><a href="/asyoulikethis/">As You Like It</a>...</li>
        </ul>
    </li>
    <li>Tragedies
        <ul>
            <li><a href="hamlet.pdf">Hamlet</a>...</li>
        </ul>
    </li>
    <li>Histories
        <ul>
            <li><a href="mailto:henryiv@king.co.uk">email</a></li>
            <ul>
                <li>Part I</li>
                <li>Part II</li>
            </ul>
            <li><a href="http://www.shakespeare.co.uk/henryv.htm">Henry V</a></li>
            <li>Richard II</li>
        </ul>
    </li>
</ul>
```
Styling list-item levels

Without any formatting, the HTML appears as a set of bulleted items arranged in the usual way. Suppose we want only top-level items to be arranged horizontally:

```
.horizontal {
    float: left;
    list-style: none;
    margin: 10px;
}
```

The `horizontal` class floats the element to the left of the one following it, removes the bullet if it’s a list item, and adds a 10 pixel margin on all sides of it.
Styling the top-level items

Rather than adding the `horizontal` class in our HTML, we add it to top-level list items only with jQuery. The `$(document).ready()` wrapper runs the code once the DOM has loaded.

```javascript
$(document).ready(function () {
  $('#selected-plays > li').addClass('horizontal');
});
```

The selector uses the **child combinator** (`>` ) to add the `horizontal` class to all top-level items only. It says, in effect:

- Find each list item (`li`)
- that is a child (`>`)
- of the element with an ID of `selected-plays` (`#selected-plays`).
Styling lower-level items

To style the rest of the list items, we can use the negation pseudo-class to identify all list items that do not have a class of horizontal:

```javascript
$(document).ready(function () {
  $('#selected-plays > li').addClass('horizontal');
  $('#selected-plays li:not(.horizontal)').addClass('sub-level');
});
```

This time we are selecting every list item (li) that:

- Is a descendant of the element with an ID of `selected-plays` (#selected-plays)
- Does not have a class of `horizontal` (:not(.horizontal))

The sub-level class gives nested items a shaded background because of another rule in the stylesheet.
Attribute selectors let us specify an element by one of its HTML properties, such as a link’s title attribute or an image’s alt attribute. For example, to select all images that have an alt attribute, we write the following:

```javascript
$('img[alt]')
```

Attribute selectors accept a wildcard syntax inspired by regular expressions for identifying:

- the value at the beginning (^) of a string
- the value at the ending ($) of a string
- a value at an arbitrary position (*) in a string
- a value that is not (!) contained in the string
Styling links

Suppose we have a stylesheet to style different classes of links:

```
a { color: #00c; }

a.mailto { background: url(images/mail.png) no-repeat right top;
            padding-right: 18px; }

a.pdflink { background: url(images/pdf.png) no-repeat right top;
            padding-right: 18px; }

a.henrylink { background-color: #fff;
               padding: 2px;
               border: 1px solid #000; }
```

Then we add the three classes—mailto, pdflink, and henrylink—to the appropriate links using jQuery.
Styling mailto links

To add a class for all email links, we construct a selector that looks for all anchor elements (a) with an href attribute ([href] that begins with mailto: (^=mailto:]):

```javascript
$(document).ready(function () {
    $('a[href ^= mailto:]').addClass('mailto');
});
```

To add a class for all links to PDF files, we use the dollar sign rather than the caret symbol. This is because we are selecting links with an href attribute that ends with .pdf:

```javascript
$(document).ready(function () {
    $('a[href ^= mailto:]').addClass('mailto');
    $('a[href$ = .pdf]').addClass('pdflink');
});
```
Combining attribute selectors

Attribute selectors can be combined as well. We can add a `henrylink` class for all links with an `href` value that both starts with `http` and contains `henry` anywhere:

```javascript
$(document).ready(function () {
  $('a[href^=mailto:]').addClass('mailto');
  $('a[href$=.pdf]').addClass('pdflink');
  $('a[href^=http][href*=henry]').addClass('henrylink');
});
```

Now the page shows a PDF icon to the right of the *Hamlet* link, an envelope next to the email link, and a white background with black border around the *Henry V* link.
jQuery adds its own custom selectors to those offered by CSS. These are mostly for picking specific elements out of a group. The syntax is the same as for CSS pseudo-classes, where the selector starts with a colon (:). For example, to select the second item from a set of div elements that all have the class horizontal, we can write:

\[
\$('div.horizontal:eq(1)'\)
\]

Note that JavaScript array numbering is **zero-based**. In contrast, CSS is **one-based**, so a CSS selector such as:

\[
\$('div:nth-child(1)'\)
\]

would select all div elements that are the first child of their respective parents. For this particular case, we could also use:

\[
\$('div:first-child'\)
\]
Styling alternate rows

Two useful custom selectors are :odd and :even. Consider this table:

```html
<table>
  <tr><td>As You Like It</td><td>Comedy</td></tr>
  <tr><td>All’s Well that Ends Well</td><td>Comedy</td><td>1601</td></tr>
  <tr><td>Hamlet</td><td>Tragedy</td><td>1604</td></tr>
  <tr><td>Macbeth</td><td>Tragedy</td><td>1606</td></tr>
  <tr><td>Romeo and Juliet</td><td>Tragedy</td><td>1595</td></tr>
  <tr><td>Henry IV, Part I</td><td>History</td><td>1596</td></tr>
  <tr><td>Henry V</td><td>History</td><td>1599</td></tr>
</table>
```

with this CSS:

```css
tr { background-color: #fff; }
.alt { background-color: #ccc; }
```

Note that in CSS, a class rule overrides an tag name rule.
Styling alternate rows

We attach the \texttt{alt} class to the even-numbered table rows:

\begin{verbatim}
$(document.ready(function () {
    $('tr:odd').addClass('alt');
}));
\end{verbatim}

Why use the \texttt{:odd} select for even-numbered rows? As a jQuery custom selector, \texttt{:odd} uses zero-based numbering.

This code considers all \texttt{tr} tags on the page; it does not “start over” for each new table. Since this table has an odd number of rows, the next table on the page will start with the \texttt{alt} coloring scheme. We could use \texttt{:nth-child} instead, which can take \texttt{even} or \texttt{odd} as its argument. \texttt{:nth-child} is the only jQuery selector that is one-based:

\begin{verbatim}
$(document.ready(function () {
    $('tr:nth-child(even)').addClass('alt');
}));
\end{verbatim}
Styling based on contents

Suppose we want to highlight any table cell that referred to one of the *Henry* plays. After adding a stylesheet rule:

```
.highlight { font-weight: bold; font-style: italics; }
```

we can add another class to all matching table cells:

```
$(document.ready(function () {
  $('tr:nth-child(even)').addClass('alt ');
  $('td:contains(Henry)').addClass('highlight ');
}));
```

The :contains() selector is case sensitive. Using:

```
$(‘td:contains(henry)’).addClass(‘highlight’)
```

would not select any cells.
jQuery has custom selectors to make it easy to work with forms:

<table>
<thead>
<tr>
<th>Selector</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>:text, :checkbox, :radio,</td>
<td>Input elements with a type attribute equal to the selector name (excluding the colon). For example :text selects &lt;input type=&quot;text&quot;&gt;</td>
</tr>
<tr>
<td>:image, :submit, :reset,</td>
<td></td>
</tr>
<tr>
<td>:password, :file</td>
<td></td>
</tr>
<tr>
<td>:input</td>
<td>Input, textarea, select, and button elements</td>
</tr>
<tr>
<td>:button</td>
<td>Button elements and input elements with a type attribute equal to button</td>
</tr>
<tr>
<td>:enabled</td>
<td>Form elements that are enabled</td>
</tr>
<tr>
<td>:disabled</td>
<td>Form elements that are disabled</td>
</tr>
<tr>
<td>:checked</td>
<td>Radio buttons/checkboxes, checked</td>
</tr>
<tr>
<td>:selected</td>
<td>Option elements that are selected</td>
</tr>
</tbody>
</table>
Form selectors

As with other selectors, form selectors can be combined to make a query more specific:

- To select all check radio buttons (but not checkboxes):
  
  $$\$(\'\:radio:checked\ʼ)$$

- To select all password inputs and disabled text inputs:
  
  $$\$(\ʼ\:password, \:text:disabled\ʼ)$$

Custom selectors use the same basic principles of CSS to build the list of matched elements.
DOM traversal methods

The selectors we have seen so far apply based on an element’s type and location within the DOM tree. There are additional DOM traversal methods to help us maneuver around the page in different ways, e.g., to move up, down, left, and right around the tree, or to specify elements in more flexible ways. Consider this query:

```
$('tr').filter(':odd').addClass('alt');
```

Though equivalent to `$('tr:odd')`, this works in a different way. All table rows are gathered, and then the resulting set of elements is examined a second time to eliminate every other item.
**DOM traversal methods**

`.filter()` can take a function as its argument, and the function is invoked for each element to decide if it should be included or not:

```javascript
$('a').filter(function () {
    return this.hostname && this.hostname !== location.hostname;
}).addClass('external');
```

**Implicit iteration** in jQuery means that the filter function will be called for each element selected by the `$('a')` query, and likewise the `addClass()` action will happen for every element in the result set. The function selects based on two requirements:

- Elements must have an `href` attribute with a domain name (`this.hostname`); this excludes `mailto` links, etc.
- The linked domain name must not match that of the current page (`location.hostname`).
Styling specific cells

Earlier, we styled all cells containing the text “Henry”:

\[
$('td:contains(Henry)').addClass('highlight');
\]

We can modify this to select the cell immediately after each cell containing “Henry”:

\[
$('td:contains(Henry)').next().addClass('highlight');
\]

Note that `next()` is applied individually to each matching element. The result is the set of all elements that immediately follow an element containing the text “Henry”, within the same table row (or other parent container).
Styling specific cells

This selects only the cell immediately following the one containing “Henry”. We could also highlight all cells following it:

\[
\$\left(\text{'td:contains(Henry)'}\right)\text{.nextAll().addClass('highlight')};
\]

If we want to include the original cell:

\[
\$\left(\text{'td:contains(Henry)'}\right)\text{.nextAll().andSelf().addClass('highlight')};
\]

Here is one way to select every element in the row:

\[
\$\left(\text{'td:contains(Henry)'}\right)\text{.parent().children().addClass('highlight')};
\]

This version finds the elements with “Henry”, then finds the parent of each such element, then walks down to find all the children of those parent elements. A number of alternative queries could achieve the same thing.
Chaining is a trick used throughout jQuery. Whenever you invoke an action on a set of elements, the same set (as a jQuery object) is returned, allowing you to immediately invoke another action on the same set of elements, or modify the set by issuing more selection/traversal methods.

Chaining allows you to:

- Select multiple sets of elements and
- do multiple things with them,

all within a single line of code. Chaining

- helps keep jQuery code concise, and it also
- improves a script’s performance when the alternative is to re-specify a selector.
Chaining

Chains of methods can be written crammed together:

```
$('td:contains(Henry)').parent()
  .find('td:eq(1)').addClass('highlight').end()
  .find('td:eq(2)').addClass('highlight');
```

Or you can add whitespace to make it easier to follow:

```
$('td:contains(Henry)') // Find every cell containing "Henry"
  .parent() // Select its parent
  .find('td:eq(1)') // Find the 2nd descendant cell
    .addClass('highlight') // Add the "highlight" class
  .end() // Return to the parent of "Henry" cell
  .find('td:eq(2)') // Find the 3rd descendant cell
    .addClass('highlight'); // Add the "highlight" class
```

This example could be achieved with simpler queries, but it demonstrates the idea behind chaining, plus it shows the `end()` method. Always use some grouping or indentation to make complex chains easier to follow.
Accessing DOM elements

Every selector expression and most other jQuery methods return a jQuery object. This is usually what we want, but sometimes we need access to the raw DOM objects. For example, you may need a DOM element to pass to another library, or you may need access to a property (such as the tag name) not accessible through the jQuery object.

The `.get()` method retrieves a DOM object from the set represented by a jQuery object. `.get(0)` returns the first DOM element in the set, etc.:

```javascript
// get the tag name of the element with ID my-element
var myTag = $('#my-element').get(0).tagName;
```

We can also use square brackets to achieve the same thing. In this way, jQuery makes the object look like an array of DOM elements:

```javascript
var myTag = $('#my-element')[0].tagName;
```