Interactive Web Development

Events

Dr Russ Ross

Dixie State University—Computer and Information Technologies

Spring 2015
**Reading:** Learning jQuery 1.3, Chapter 3
Performing tasks on page load

Register code to be executed after the page is loaded using:

```javascript
$(document).ready(function () {
    // do stuff here
    $('#main').hide();
});
```

This differs from the `window.onload` event in its timing:

- `onload` waits until the complete page is ready, including all downloaded objects.
- `ready` fires when the DOM is ready, but images and other linked objects may not have finished downloading.

This is usually the best way, but attributes like image sizes may not be available. Use `$(document).load(function () { ... })`; if you need the regular `onload` event.
Multiple scripts on one page

Consider registering multiple handlers for a single event:

```javascript
function doStuff() { ... }
function doMoreStuff() { ... }
window.onload = doStuff; // note: not doStuff()
window.onload = doMoreStuff;
```

This fails because the second assignment overwrites the first.

Using jQuery to register handlers puts them in a queue:

```javascript
$(document).ready(doStuff);
$(document).ready(doMoreStuff);
```

Do not mix and match how you register handlers: anything registered with jQuery will clobber anything you had registered before, and registering an event by hand will clobber anything jQuery has registered.
Shortcuts for code brevity

When called with no arguments, the jQuery factory function uses `document` as a default, so:

```javascript
$(document).ready(function () {
    // do stuff
});
```

is the same as:

```javascript
$(function () {
    // do stuff
});
```

and when you give a function to the factory method, it makes the call to `ready`, so you can use:

```javascript
$(function () {
    // do stuff
});
```
Coexisting with other libraries

Everyone uses $() as a shortcut function. You can use jQuery() instead, but that is inconvenient. The ready function has a trick to work around this. It passes the jQuery object as a parameter to its callback function, so you can use:

```javascript
jQuery(document).ready(function ($) {
    // in here you can use $ for jQuery
});
```

It works with the shortcut, too:

```javascript
jQuery(function ($) {
    // this can use $
});
```

Since most of your jQuery code will be written in this callback, you can conveniently use $(), without needing the global binding of $.
A simple style switcher

A simple style switcher: the user clicks on buttons to change the page styling. The HTML:

```html
<div id="switcher">
  <h3>Style Switcher</h3>
  <div class="button selected" id="switcher-default">
    Default
  </div>
  <div class="button" id="switcher-narrow">
    Narrow Column
  </div>
  <div class="button" id="switcher-large">
    Large Print
  </div>
</div>
```
We start with the “Large Print” button. First the CSS:

```css
body.large .chapter {
    font-size: 1.5em;
}
```

Our goal is to execute this when the button is clicked:

```javascript
$('body').addClass('large');
```

To install the event handler, use the `bind()` method:

```javascript
$(function () {
    $('#switcher-large').bind('click', function () {
        $('body').addClass('large');
    });
});
```
Enabling the other buttons

We can use similar code to enable all three buttons:

```javascript
$(function () {
    $('#switcher-default').bind('click', function () {
        $('body').removeClass('narrow');
        $('body').removeClass('large');
    });
    $('#switcher-narrow').bind('click', function () {
        $('body').addClass('narrow');
        $('body').removeClass('large');
    });
    $('#switcher-large').bind('click', function () {
        $('body').removeClass('narrow');
        $('body').addClass('large');
    });
});
```

This requires another CSS rule:

```css
body.narrow .chapter { width: 400px; }
```
To show the user which style is selected, we’ll add the `.selected` class, which bolds the label:

```css
.selected {  
  font-weight: bold;  
}
```

We can use the `context` of the event to apply it to the correct button:

```javascript
$(this).addClass('selected');
```

after removing it from all of the buttons using implicit iteration:

```javascript
$('#switcher .button').removeClass('selected');
```
The full code

```javascript
$(function () {
  $('#switcher-default').bind('click', function () {
    $('body').removeClass('narrow');
    $('body').removeClass('large');
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
  $('#switcher-narrow').bind('click', function () {
    $('body').addClass('narrow');
    $('body').removeClass('large');
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
  $('#switcher-large').bind('click', function () {
    $('body').removeClass('narrow');
    $('body').addClass('large');
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```
We can factor out the common code into another handler and use chaining to improve the code:

```javascript
$(function () {
  $('#switcher-default').bind('click', function () {
    $('body').removeClass('narrow').removeClass('large');
  });
  $('#switcher-narrow').bind('click', function () {
    $('body').addClass('narrow').removeClass('large');
  });
  $('#switcher-large').bind('click', function () {
    $('body').removeClass('narrow').addClass('large');
  });
  $('#switcher .button').bind('click', function () {
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```
Improving the code

With no arguments, `removeClass()` removes all classes:

```javascript
$(function () {
    $('#switcher-default').bind('click', function () {
        $('body').removeClass();
    });
    $('#switcher-narrow').bind('click', function () {
        $('body').removeClass().addClass('narrow');
    });
    $('#switcher-large').bind('click', function () {
        $('body').removeClass().addClass('large');
    });
    $('#switcher .button').bind('click', function () {
        $('#switcher .button').removeClass('selected');
        $(this).addClass('selected');
    });
});
```
Improving the code

Events are triggered in the order they are bound, so we can factor this further:

```javascript
$(function () {
  $('#switcher .button').bind('click', function () {
    $('body').removeClass();
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
  $('#switcher-narrow').bind('click', function () {
    $('body').addClass('narrow');
  });
  $('#switcher-large').bind('click', function () {
    $('body').addClass('large');
  });
});
```
Finally, we can do it all in a single handler by taking further advantage of the event context:

```javascript
$(function () {
    $('#switcher .button').bind('click', function () {
        $('body').removeClass();
        if (this.id == 'switcher-narrow') {
            $('body').addClass('narrow');
        } else if (this.id == 'switcher-large') {
            $('body').addClass('large');
        }
        $('#switcher .button').removeClass('selected');
        $(this).addClass('selected');
    });
});
```
Most common events have shorthand binding events. For example:

```javascript
$(function () {
  $('#switcher .button').click(function () {
    $('body').removeClass();
    if (this.id == 'switcher-narrow') {
      $('body').addClass('narrow');
    } else if (this.id == 'switcher-large') {
      $('body').addClass('large');
    }
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```
Shorthand events

Shorthand event methods exist for all standard DOM events:

- `blur`
- `change`
- `click`
- `dblclick`
- `error`
- `focus`
- `keydown`
- `keypress`
- `keyup`
- `load`
- `mousedown`
- `mousemove`
- `mouseout`
- `mouseover`
- `mouseup`
- `resize`
- `scroll`
- `select`
- `submit`
- `unload`

In addition to the events directly supported by the browser, jQuery offers compound events for convenience, including:

- `ready`
- `toggle`
- `hover`
Showing and hiding advanced features

To switch between having the buttons visible and hidden, we need to alternate between two states. Rather than implementing this manually, we can use the pseudo-event. First the CSS:

```css
.hidden {
    display: none;
}
```

And the code:

```javascript
$(function () {
    $('#switcher h3').toggle(function () {
        $('#switcher .button').addClass('hidden');
    }, function () {
        $('#switcher .button').removeClass('hidden');
    });
});
```

toggle alternates between the handlers each time it is clicked (note: you can have more than two).
Showing and hiding advanced features

Toggling a class is common enough that it has a special method:

```
$(function () {
    $('#switcher h3').click(function () {
        $('#switcher button').toggleClass('hidden');
    });
});
```

Note the use of implicit iteration to handle all of the buttons in one step.
Highlighting clickable items

Best to mark clickable elements somehow:

```css
#switcher .hover {  
cursor: pointer;  
background-color: #afa;
}
```

CSS supports a :hover pseudo-class, but IE support for it is restricted. Instead, jQuery offers a pseudo-event:

```javascript
$(function () {
  $('#switcher .button').hover(function () {
    $(this).addClass('hover');
  }, function () {
    $(this).removeClass('hover');
  });
});
```

The first function is called when the mouse enters the element, the second when it leaves.
Consider a hierarchy of DOM objects:

```
<div class="foo">
  <span class="bar">
    <a href="http://www.example.com/">
      The quick brown fox jumps over the lazy dog.
    </a>
  </span>
  <p>
    How razorback-jumping frogs can level six piqued gymnasts!
  </p>
</div>
```

When the user clicks on the link, the click could be intended for several elements: the `<a>`, the `<span>`, and the `<div>` have all been clicked. Only the `<p>` is not part of the interaction.
Event propagation

Two approaches:

- Event capturing: the event is given to the top-level element first, then to more and more specific elements. In this example, the `<div>`, then the `<span>`, then the `<a>`.

- Event bubbling: the event is given to the most specific element first, then to higher- and higher-level elements. In this example, the `<a>`, then the `<span>`, then the `<div>`.

Browser support is mixed; jQuery always registers events to be handled using event bubbling.
Side effects of event bubbling

Event bubbling can have subtle consequences:

- The mouse leaves the `<a>` element and a `mouseout` event is fired.
- The event bubbles up to the `<span>` and then to the `<div>`.
- The `<div>` is watching for `mouseout` events. It is only interested in those generated by the mouse leaving the `<div>`, but gets events for the mouse leaving one of its descendent elements.

jQuery makes some of this easier: `hover()` is aware of bubbling and does “the right thing”.

Usually `hover` is the right thing, but if you need `mouseover` or `mouseout` but not both, it also offers `mouseenter` and `mouseleave` pseudo-events, which act the same as `mouseover` and `mouseout`, respectively, but circumventing bubbling concerns.
The event object

An example of where we must handle bubbling concerns by hand arises if we change out code to allow clicking anywhere in the `<div>` to collapse/expand the buttons:

```javascript
$(function () {
  $('#switcher').click(function () {
    $('#switcher .button').toggleClass('hidden');
  });
});
```

This works, except that clicking one of the buttons also collapses the entire `<div>`. 
Event targets

To fix this we need the event object:

```javascript
$(function () {
    $('#switcher').click(function (event) {
        $('#switcher .button').toggleClass('hidden');
    });
});
```

jQuery gives us `event.target` even in browsers that do not support it directly:

```javascript
$(function () {
    $('#switcher').click(function (event) {
        if (event.target == this) {
            $('#switcher .button').toggleClass('hidden');
        }
    });
});
```

Now the click is ignored if it was on a sub-element.
Stopping event propagation

We have gone too far: now clicking the `<h3>` fails to collapse the `<div>`.

In this case, we want to stop the event from propagating when it is handled by a button, but catch it in every other case:

```javascript
$(function () {
    $('#switcher .button').click(function (event) {
        $('body').removeClass();
        if (this.id == 'switcher-narrow') {
            $('body').addClass('narrow');
        } else if (this.id == 'switcher-large') {
            $('body').addClass('large');
        }
        $('#switcher .button').removeClass('selected');
        $(this).addClass('selected');
        event.stopPropagation();
    });
});
```

This is why bubbling up is generally the better way to implement events: give more specific elements the first chance to respond.
Default actions

Sometimes `stopPropagation()` is not enough: if the element was a `<a>`, the browser would normally follow the link when clicked.

`stopPropagation()` prevents enclosing elements from receiving the event, but does not disable the default action. For that, call `event.preventDefault()`.

Returning `false` from the event handler is the same as calling both `stopPropagation()` and `preventDefault()`.
Event delegation

To avoid installing many event handlers, we can capture the event in a higher-level element and check if it applies to one of the buttons:

```javascript
$(function () {
    $('#switcher').click(function (event) {
        if ($(event.target).is('.button')) {
            $('body').removeClass();
            if (this.id == 'switcher-narrow') {
                $('body').addClass('narrow');
            } else if (this.id == 'switcher-large') {
                $('body').addClass('large');
            }
        } else if (this.id == 'switcher-narrow') {
            $(this).addClass('selected');
            event.stopPropagation();
        }
    });
});
```

`is()` checks the currently selected elements against a selector expression, returning `true` if at least one element matches.
Event delegation

Notes:

- **this** now refers to the `<div id="switcher">` (since that is where the event handler was installed), so use `event.target` to get at the specific button element.

- `.hasClass()` would also work in this case; `.is()` is more general.

- jQuery 1.3 and later have `on()` and `off()`, which act the same as `bind()` and `unbind()`, but are implemented using event delegation instead of direct binding. This means they can be triggered by elements that do not yet exist.

For details, see:

http://api.jquery.com/on
Event delegation

The side-effect is back: clicking a button collapses the switcher `<div>` again. The handlers are bound to the same element, so `.stopPropagation()` does not prevent the collapser handler from being invoked. Another `.is()` check can fix it:

```javascript
$(function () {
  $('#switcher').click(function (event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  });
});
```
Removing an event handler

To prevent collapsing the switcher box when in a non-default style, we can unbind the event handler:

```javascript
$(function () {
  $('#switcher').click(function (event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  });
  $('#switcher-narrow, #switcher-large').click(function () {
    $('#switcher').unbind('click');
  });
});
```

Problem: the buttons will no longer work, because all click handlers were removed.
Event namespacing

If we return to the non-shorthand method of binding event handlers, we can categorize our event handlers:

```javascript
$(function () {
    $('#switcher').bind('click .collapse', function (event) {
        if (!$(event.target).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    });
    $('#switcher-narrow, #switcher-large').click(function () {
        $('#switcher').unbind('click .collapse');
    });
});
```

Now we have a name for the event handler (or handlers—multiple handlers can be registered with the same suffix name), so we can remove it (them) without disturbing other handlers.

The .collapse suffix is ignored by the event handling system.
Rebinding events

To bind the handler again when we switch back to a default style, we need a name for the handler:

```
$(function () {
    var toggleStyleSwitcher = function (event) {
        if (!$(event.target).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    }
    $('#switcher').bind('click.collapse', toggleStyleSwitcher);
});
```

Note: toggleStyleSwitcher, not toggleStyleSwitcher().
Rebinding events

Now we can rebind it later without repeating the definition:

```javascript
$(function () {
    var toggleStyleSwitcher = function (event) {
        if (!$(event.target).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    };
    $('#switcher').bind('click .collapse', toggleStyleSwitcher);

    $('#switcher-narrow, #switcher-large').click(function () {
        $('#switcher').unbind('click .collapse');
    });
    $('#switcher-default').click(function () {
        $('#switcher').bind('click .collapse', toggleStyleSwitcher);
    });
});
```

Note: Clicking the “normal” button repeatedly would bind the handler multiple times. jQuery will only bind the *same* handler once. Multiple anonymous functions from the same code are *not* the same function.

Dr Russ Ross  (Dixie State University)
Rebinding events

You can unbind a named function without using namespaces:

```javascript
$(function () {
    var toggleStyleSwitcher = function (event) {
        if (!$($(event.target)).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    }
    $('#switcher').click(toggleStyleSwitcher);

    $('#switcher-narrow, #switcher-large').click(function () {
        $('#switcher').unbind('click', toggleStyleSwitcher);
    });
    $('#switcher-default').click(function () {
        $('#switcher').click(toggleStyleSwitcher);
    });
});
```

To unbind a handler after the first time it is triggered, use `one()`:

```javascript
$('#switcher').one('click', toggleStyleSwitcher);
```
Simulating user interaction

Sometimes it is convenient to trigger a handler directly from your code.

To start with the switcher collapsed, we could simulate a user clicking on the `<div>`:

```
$(function () {
    $('#switcher').trigger('click');
});
```

The `trigger()` method has the same shortcuts as `bind()`. When called with no arguments, they trigger the action rather than binding it:

```
$(function () {
    $('#switcher').click();
});
```
Keyboard events

Keyboard events are a bit different than mouse events:

- There are two kinds of events: `keyup` and `keydown` react to the keyboard directly. If you want to know which key the user pressed, use these.

- `keypress` reacts to text input. If you want to know which character ended up on the screen, use this.

- The target of a key event is the element that currently has keyboard focus. Not all elements can have focus, mostly keyboard-driven elements such as form fields.

- Event bubbling still works, so we can capture keys from any focus by listening on `document`.

- To know which key was pressed, examine `event.keyCode`. 
Keyboard events

For alphabetic keys, `event.keyCode` is the ASCII code of the uppercase letter.

```javascript
$(function () {
  $(document).keyup(function (event) {
    switch (String.fromCharCode(event.keyCode)) {
      case 'D':
        $('#switcher-default').click();
        break;
      case 'N':
        $('#switcher-narrow').click();
        break;
      case 'L':
        $('#switcher-large').click();
        break;
    }
  });
});
```