

jQuery Fundamentals Training

Events and Event Handlers

Lesson 1, Activity 2: jQuery provides simple methods for attaching event handlers to selections. When an event occurs, the provided function is executed. Inside the function, `this` refers to the element that was clicked.

For details on jQuery events, visit <http://api.jquery.com/category/events/>.

The event handling function can receive an event object. This object can be used to determine the nature of the event, and to prevent the event's default behavior.

For details on the event object, visit <http://api.jquery.com/category/events/event-object/>.

Connecting Events to Elements

jQuery offers convenience methods for most common events, and these are the methods you will see used most often. These methods, including `$.fn.click`, `$.fn.focus`, `$.fn.blur`, `$.fn.change`, etc, are shorthand for jQuery's `$.fn.bind` method. The `bind` method is useful for binding the few browser events for which jQuery does not have a convenience method, or binding the same handler function to multiple events, and is also used when you are working with custom events.

Within the event handler function, the `this` object will be the element that you bound the event handler to (unless you use `$.proxy` to change that).

Event Binding Using a Convenience Method

Code Sample:

[jqy-events/Demos/convenience-event-binding.html](#)

```
<html>
<head>
<script src="../../jqy-lib/jquery.js"></script>
<script src="../../jqy-lib/fix-console.js"></script>
</head>
<body>
<p>One</p>
<p>Two</p>
<p>Three</p>
<p>Four</p>
<script>

// Using this as an ordinary HTML element object
$('p:odd').click(function() {
    console.log('click odd' + ": " + this.innerHTML);
});

// Using $(this) as a jQuery single-element collection
$('p:even').click(function() {
    console.log('click even' + " - " + $(this).html());
});
</script>
</body>
</html>
```

We establish two event handlers: one for odd rows, and the other for even rows. For odd rows, we use the `this` object, which will be the object that received the click (the paragraph element), as an ordinary JavaScript HTML element object, and access its `innerHTML` property.

For even rows, we wrap `this` into a jQuery collection object with `$(this)`, and then use the `html()` method for the same purpose.

Event Binding Using the `$.fn.bind` Method

This method can be used to bind a handler for any type of event, including custom events that you would create yourself.

```
$('p').bind('click', function() { console.log('click'); });
```

Event Binding With Data

Each event-binding convenience function has a second form that takes two parameters. In this form, the second parameter is the target, and the first parameter is a map of data that will be given to the handler through the event object.

Code Sample:

[jqy-events/Demos/data-event-binding.html](#)

```
<html>
<head>
<script src="../../../jqy-lib/jquery.js"></script>
</head>
<body>
<p>One</p>
<p>Two</p>
<p>Three</p>
<p>Four</p>
<script>
var counter = 0;

// Using this as an ordinary HTML element object
$('p:odd').click(
  { type: 'odd' },
  function(e) {
    console.log('click ' + e.data["type"] + ": " + this.innerHTML);
  }
);

// Using $(this) as a jQuery single-element collection
$('p:even').click(
  { type: 'even' },
  function(e) {
    console.log('click ' + e.data["type"] + " - " + $(this).html());
  }
);
```

```
);
</script>
</body>
</html>
```

We again establish two event handlers, but for each one have added a data object with a type property of either "odd" or "even". When the event occurs, we can retrieve that information through the `data` property of the event object passed to the handler.

The `bind` method also has a form that accepts a data object, as the second parameter (between the event name and the handler function).

Connecting Events to Run Only Once

Sometimes you need a particular handler to run only once -- after that, you may want no handler to run, or you may want a different handler to run. jQuery provides the `$.fn.one` method for this purpose.

Switching Handlers Using the `$.fn.one` Method

Code Sample:

jqy-events/Demos/switching-handlers.html

```
<html>
<head>
<script src="../../jqy-lib/jquery.js"></script>
<script src="../../jqy-lib/fix-console.js"></script>
</head>
<body>
<p>One</p>
<p>Two</p>
<p>Three</p>
<p>Four</p>
<script>
$( 'p' ).one (
```

```

'click',
function() {
  console.log('You just clicked this for the first time!');
  $(this).click(
    function() { console.log('You have clicked this before!'); }
  );
}
);
</script>
</body>
</html>

```

The `$.fn.one` method is especially useful if you need to do some complicated setup the first time an element is clicked, but not subsequent times.

Code Sample:

[jqy-events/Demos/lazy-load.html](#)

```

---- C O D E   O M I T T E D ----
</div>
<div class="disclosure">
  <h3 class="control">Edit Content</h3>
  <div class="content"></div>
</div>

<script src="../../jqy-libs/jquery.js"></script>
<script src="../../jqy-libs/fix-console.js"></script>
<script>
$(document).ready(
function() {
  $('<div class="disclosure"> .control').each(function() {
    var $this = $(this);
    var $div = $this.next('div.content');
    $this.one('click',
      function(e) {
        e.preventDefault();
        $div.html("<p>Some</p><p>New</p><p>Content</p>");
        $div.slideDown();
        $this.click(
          function() {
            $div.slideToggle();
          }
        );
      }
    );
  });
}
);
}

```

```

    );
  });
}
);
</script>
</body>
</html>

```

In this application, there is an initially empty, hidden, div beneath a trigger element. Clicking on the trigger will run a one-time event handler to:

- load the div (which in real life would be done using Ajax)
- show the div
- set up ongoing event handling to toggle the open/close state of the div with each click on the trigger

Disconnecting Events

To disconnect an event handler, you use the `$.fn.unbind` method and pass in the event type to unbind. If you attached a named function to the event, then you can isolate the unbinding to that named function by passing it as the second argument.

Unbinding All Click Handlers on a Selection

```

$('p').unbind('click');

```

Unbinding a Particular Click Handler

```

var foo = function() { console.log('foo'); };
var bar = function() { console.log('bar'); };

$('p').bind('click', foo).bind('click', bar);
$('p').unbind('click', bar); // foo is still bound to the click event

```

Namespacing Events

For complex applications and for plugins you share with others, it can be useful to namespace your events so you don't unintentionally disconnect events that you didn't or couldn't know about.

```
$('p').bind('click.myNamespace', function() { /* ... */ });  
$('p').unbind('click.myNamespace');  
$('p').unbind('.myNamespace'); // unbind all events in the namespace
```


Lesson 1, Activity 4: Inside the Event Handling Function

As mentioned in the overview, the event handling function receives an event object, which contains many properties and methods. The event object is most commonly used to prevent the default action of the event via the `preventDefault` method. However, the event object contains a number of other useful properties and methods, including:

- `pageX`, `pageY` - The mouse position at the time the event occurred, relative to the top left of the page.
- `type` - The type of the event (e.g. "click").
- `which` - The button or key that was pressed.
- `data` - Any data that was passed in when the event was bound.
- `target` - The DOM element that initiated the event.
- `preventDefault()` - Prevent the default action of the event (e.g. following a link).
- `stopPropagation()` - Stop the event from bubbling up to other elements.

In addition to the event object, the event handling function also has access to the DOM element that the handler was bound to via the keyword `this`. To turn the DOM element into a jQuery object that we can use jQuery methods on, we simply do `$(this)`, often following this idiom:

```
var $this = $(this);
```

Using `event.preventDefault()`

Code Sample:

jqy-events/Demos/prevent-link-following.html

```
<html>
<head>
<script src="../../jqy-libs/jquery.js"></script>
</head>
<body>
<p><a href="http://www.webucator.com/">One</a></p>
<p><a href="bind-event-binding.html">Two</a></p>
<p>Three</p>
<p>Four</p>
<script>
$('a').click(function(e) {
    var $this = $(this);
    if ($this.attr('href').match('http://')) {
        e.preventDefault();
    }
});
</script>
</body>
</html>
```

Lesson 1, Activity 5: Triggering Event Handlers

jQuery provides a way to trigger the event handlers bound to an element without any user interaction via the `$.fn.trigger` method. While this method has its uses, it should not be used simply to call a function that was bound as a click handler. Instead, you should store the function you want to call in a variable, and pass the variable name when you do your binding. Then, you can call the function itself whenever you want, without the need for `$.fn.trigger`.

Also, for the convenience methods, invoking them with no parameters triggers the event on the selected elements.

```
// A not-so-good practice
var foo = function(e) {
  console.log(e);
};
$('p').click(foo);
$('p').trigger('click');

// a better way - look for the event object
// if not present, means that call was not event-generated
var foo = function(e) {
  if (e) {
    console.log(e);
  } else {
    console.log('this didn\'t come from an event!');
  }
};
$('p').click(foo);
foo();
```

Lesson 1, Activity 7: **Increasing Performance with Event Delegation**

You'll frequently use jQuery to add new elements to the page, and when you do, you may need to bind events to those new elements -- events you already bound to similar elements that were on the page originally. Instead of repeating your event binding every time you add elements to the page, you can use event delegation.

There are two forms of event delegation: live tracking of jQuery collections, and true delegation. With live tracking, jQuery keeps track of a query over time, and when the set of elements returned by the query changes, it will update the event binding accordingly. With this approach, the handlers are individually bound to each element matching the query. Live tracking is deprecated -- since it does bind individual handlers to each element, a large set of elements will result in an equally large set of handlers. Also, processing time will be expended tracking the query to keep the set of elements up to date. Live tracking uses

```
$.fn.live.
```

With true event delegation, you bind your event to a container element, and provide a subquery that finds the actual elements you want to handle the event for. When the event occurs, jQuery looks to see which contained element it occurred on. Event delegation uses

```
$.fn.delegate.
```

While most people discover event delegation while dealing with elements added to the page later, it has some performance benefits even if you never add more elements to the page. The time required to bind

event handlers to hundreds of individual elements is non-trivial; if you have a large set of elements, you should consider delegating related events to a container element.

Event Delegation Using `$.fn.live` and `$.fn.delegate`

Code Sample:

<jqy-events/Demos/event-delegation.html>

```

---- C O D E   O M I T T E D ----
<body>
<ul id="myUnorderedList">
  <li><a href="http://www.webucator.com/">One</a></li>
  <li><a href="bind-event-binding.html">Two</a></li>
  <li>Three</li>
  <li>Four</li>
</ul>
<ol id="myOrderedList">
  <li><a href="http://www.webucator.com/">One</a></li>
  <li><a href="bind-event-binding.html">Two</a></li>
  <li class="clickable">Three</li>
  <li>Four</li>
</ol>
<script>
$('#myUnorderedList li').live('click', function(e) {
  var $myListItem = $(this);
  console.log($myListItem.html());
});

$('#myOrderedList').delegate(
  "li.clickable",
  'click',
  function(e) {
    var $myListItem = $(this);
    console.log($myListItem.html());
  }
);

// Now add items that should also receive the events
$('#myUnorderedList')
  .append("<li>New List Item</li>");
$('#myOrderedList')
  .append("<li class='clickable'>New Clickable Item</li>")
  .append("<li>New Unclickable Item</li>");
</script>
</body>

```

</html>

Unbinding Delegated Events

If you need to remove delegated events, you can't simply unbind them. Instead, use `$.fn.undelegate` for events connected with `$.fn.delegate`, and `$.fn.die` for events connected with `$.fn.live`. As with `bind`, you can optionally pass in the name of the bound function. As of jQuery 1.6, you can also use namespaces with `delegate` and `undelegate`.

Unbinding Delegated Events

```
// unbinding events bound with $.fn.delegate
$('#myUnorderedList').undelegate('li.clickable', 'click');

// unbinding events bound with $.fn.live
$('#myUnorderedList li').die('click');
```

Lesson 1, Activity 8: Event Helpers

jQuery offers two event-related helper functions that can save you a few keystrokes.

`$.fn.hover`

The `$.fn.hover` method lets you pass one or two functions to be run when the `mouseenter` and `mouseleave` events occur on an element. If you pass one function, it will be run for both events; if you pass two functions, the first will run for `mouseenter`, and the second will run for `mouseleave`.

Note: Prior to jQuery 1.4, the `$.fn.hover` method required two functions.

`$.fn.toggle`

Much like `$.fn.hover`, the `$.fn.toggle` method receives two or more functions; each time a click event occurs, the next function in the list is called. Generally, `$.fn.toggle` is used with just two functions, but technically you can use as many as you'd like.

The `hover` and `toggle` Functions

Code Sample:

<jqy-events/Demos/hover-toggle.html>

```
<html>
<head>
<style>
.hover { background-color:#CCFFCC; }
.toggle1 { color: red; }
```

```

.toggle2 { color: green; }
.toggle3 { color: blue; }
</style>
<script src="../../jqy-lib/jquery.js"></script>
</head>
<body>
<ul id="myUnorderedList">
  <li><a href="http://www.webucator.com/">One</a></li>
  <li><a href="bind-event-binding.html">Two</a></li>
  <li>Three</li>
  <li>Four</li>
</ul>
<ol id="myOrderedList">
  <li><a href="http://www.webucator.com/">One</a></li>
  <li><a href="bind-event-binding.html">Two</a></li>
  <li class="clickable">Three</li>
  <li>Four</li>
</ol>
<script>

$('ul li').hover(function() {
    $(this).toggleClass('hover');
});

$('ol li').hover(
    function() {
        $(this).addClass('hover');
    },
    function() {
        $(this).removeClass('hover');
    }
);

$('li').toggle(
    function() {
        $(this).addClass('toggle1');
        $(this).removeClass('toggle2');
        $(this).removeClass('toggle3');
    },
    function() {
        $(this).addClass('toggle2');
        $(this).removeClass('toggle1');
        $(this).removeClass('toggle3');
    },
    function() {
        $(this).addClass('toggle3');
        $(this).removeClass('toggle2');
        $(this).removeClass('toggle1');
    },
    function() {
        $(this).removeClass('toggle3');
    }
);

```



```
$(this).removeClass('toggle2');  
$(this).removeClass('toggle1');  
}  
);  
  
</script>  
</body>  
</html>
```

Lesson 1, Activity 9: Create an Input Hint

Duration: 15 to 30 minutes.

Open the file [ClassFiles/jqy-events/Exercises/index.html](#) in your browser. Use the file [ClassFiles/jqy-events/Exercises/js/inputHint.js](#). Your task is to use the text of the label for the search input to create "hint" text for the search input.

The concept here is that the page loads in a usable fashion, but, if the user has JavaScript enabled, we can improve their experience. If the user does not have JavaScript, the page still works acceptably.

1. Set the value of the search input to the text of the `label` element.
2. Add a class of `hint` to the search input.
3. Remove the `label` element from the page (since our script now handles the hint, we will not need this element any more).
4. Bind a focus event to the search input that removes the hint text and the `hint` class.
5. Bind a blur event to the search input that restores the hint text and `hint` class.

Challenge

It's somewhat annoying that any text the user enters goes away when they leave the box. It would be nicer if we only replaced the value if it is different from the hint, or if the box is empty. As a challenge, revise the code so that the hint is only performed if the box is empty, and that the box is cleared only if the current text is the hint text.

As a second challenge, how could you set up a scenario where there were multiple text boxes in a form, and some subset of those should receive this treatment?

Solution:

jqy-events/Solutions/js/inputHint.js

```
$(document).ready(function() {
    var $search = $('#search');
    var $input = $search.find('input.input_text');
    var hint = $search.find('label').remove().text();

    $input
        .val(hint)
        .addClass('hint')
        .focus(function() { $(this).val('').removeClass('hint'); } )
        .blur(function() { $(this).val(hint).addClass('hint'); } );
});
```

Challenge 1 Solution:

jqy-events/Solutions/js/inputHint-challenge1.js

```
$(document).ready(function() {
    var $search = $('#search');
    var $input = $search.find('input.input_text');
    var hint = $search.find('label').remove().text();

    $input
        .val(hint)
        .addClass('hint')
        .focus(function() {
            var entry = $input.val();
            $input.removeClass('hint');
            if (entry == hint) $input.val('');
        })
        .blur(function() {
            if (!$.trim($input.val())) $input.val(hint);
            if ($input.val() == hint) $input.addClass('hint');
        });
    }
});
```

Challenge 2 Solution:

jqy-events/Solutions/js/inputHint-challenge2.js

```
$(document).ready(function() {
```

```

var $search = $('#search');
var $input = $search.find('input.input_text');

$input.each(function() {
    var hint = $search.find('label[for=' + this.name + ']')
        .remove()
        .text();
    var $this = $(this);

    $this
        .val(hint)
        .addClass('hint')
        .focus(function() {
            var entry = $this.val();
            $this.removeClass('hint');
            if (entry == hint) $this.val('');
        })
        .blur(function() {
            if (!$.trim($this.val())) $this.val(hint);
            if ($this.val() == hint) $this.addClass('hint');
        });
});
});

```

Challenge 2a Solution:

[jqy-events/Solutions/js/inputHint-challenge2a.js](#)

```

// Uses one event handler and data to remember associated hint text
// as opposed to an individual event with a closure for each item
$(document).ready(function() {
    var $search = $('#search');
    var $input = $search.find('input.input_text');
    $input.each(function() {
        var $label = $search.find('label[for=' + this.name + ']');
        var hint = $label.text();
        $label.remove();
        $.data(this, 'hint', hint);
        this.value = hint;
    });

    $input
        .addClass('hint')
        .focus(function(e) {
            $this = $(this);
            var entry = $this.val();
            $this.removeClass('hint');
            if (entry == $.data(this, 'hint')) $this.val('');
        })

```

```
.blur(function(e) {  
  $this = $(this);  
  if (!$.trim($this.val())) $this.val($.data(this, 'hint'));  
  if ($this.val() == $.data(this, 'hint')) $input.addClass('hint');  
});  
}  
);
```

Lesson 1, Activity 11: Add Tabbed Navigation

Duration: 20 to 40 minutes.

Continue to use [index.html](#). Use the file [jqy-events/Exercises/js/tabs.js](#).

Your task is to create tabbed navigation for the two `div.module` elements (*blog* and *specials*). To accomplish this:

1. Hide all of the modules (the `div.module` elements), but first add the `tabContent` class to them.
2. Create an unordered list element before the first module. Add the `tabBar` class to the unordered list you created.
3. Iterate over the modules using `$.fn.each`. For each module, use the text of the `h2` element as the text for a list item that you add to the unordered list element.
4. When you create each list item, bind a click event to it that:
 1. Shows the related module, and hides any other modules
 2. Adds a class of `current` to the clicked list item
 3. Removes the class `current` from the other list item(s)
 4. For the click handler to "remember" the associated module, you can either use `data`, or a local variable created in the outer function (the function supplied to `each`)
5. Add the `tabs` class to each list item
6. Finally, show the first tab. You should also ensure that the styling of the tabs matches what is showing, that is, the first tab should have the `current` class, and the others not.