Event Handling

DEVELOPMENTOR

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

Objectives

- Bind and unbind event handlers
- Handle events using the normalized Event object
- Use “live” events

.on(eventType, [eventData], handler)

- .on() adds a handler for all selected elements
  - calls addEventListener or attachEvent for each element
  - normalizes the event object so that all browsers receive consistent data in their handlers

```
$('#addButton').on('click', function(e) {
    // Handle click event here...
});
```

Binding multiple events to the same handler

- The eventType argument can be a space-delimited list of event types

```
$('#content').on('mouseenter mouseleave', function(e) {
    $(this).toggleClass('highligt');
});
```

Binding multiple events to different handlers

- .on() accepts an object mapping event types to handlers

```
$('#content').on({
    click: function(e) {
        // ...
    },
    dblclick: function(e) {
        // ...
    }
});
```

Shortcut methods

- Common events have shortcut methods available for easy event handling
  - blur change click dblclick error focus focusin focusout keydown keypress keyup load mousedown mouseenter mouseleave mousemove mouseout mouseover mouseup resize scroll select submit unload
- All shortcuts accept (optional) event data and handler

```
$('#addButton').click(function(e) {
    // Handle click event here...
});
```

Event handlers

- Event handlers invoked in the order in which they were bound
  - even in browsers that “naturally” invoke them in reverse (IE)
- Inside event handlers, “this” references the “raw” DOM element that the handler was bound to
  - use $(this) to create jQuery object for access to jQuery methods

```
$('#saveButton').click(function(e) {
    $(this).prop('disabled', true);
    // Make Ajax call...
});
```

Returning false

- Event handlers that explicitly return false cause jQuery to invoke event.preventDefault() and event.stopPropagation()
  - best not to return false and use the appropriate method

The Event object

- jQuery's Event object normalizes the difference between browser implementations
  - original event object available via event.originalEvent property
- Most properties from DOM event are copied onto jQuery's
  - with some getting "fixed"
- Optional event data specified when binding available via event.data
  - can be any type of object

Event targets

- event.target is the element that originated the event (the element that was clicked)
  - could be different from “this” if the event bubbled up the tree
- event.currentTarget is the current element handling the event
  - usually the same as “this”
  - could be different when using $.proxy()
- event.relatedTarget is only set for mouse-related events
  - for “mousein”, it’s the element being exited
  - for “mouseout”, it’s the element being entered

Key event properties

- event.keyCode
  - Unicode value of pressed key (not always set)
- event.charCode
  - Unicode value of pressed key (not always set)
- event.which
  - Unicode value of pressed key (always set—use this one!)
- event.altKey, event.ctrlKey, event.metaKey, event.shiftKey
  - true if the corresponding key is pressed

Mouse event properties

- event.pageX and event.pageY
  - mouse position relative to document
- event.clientX and event.clientY
  - mouse position relative to browser
- event.screenX and event.screenY
  - mouse position relative to screen
- event.which
  - mouse button clicked
  - 1 is left button
  - 2 is middle button
  - 3 is right button

Event object methods

- event.preventDefault() prevents browser from taking default action
  - clicking a link navigates to new URL
  - clicking submit button submits form
- event.stopPropagation() stops event from bubbling up DOM tree
- event.stopImmediatePropagation() stops other handlers from receiving event and stops event from bubbling up DOM tree

mouseenter/mouseleave

- mouseenter and mouseleave events are proprietary to Internet Explorer
  - so useful that they're simulated by jQuery
- .hover() method binds both mouseenter and mouseleave with one call

Other methods for binding events

- .one(eventType, [eventData], handler)
  - executes handler once per element
- .toggle(handler1, handler2, [handlerN])
  - executes different handlers on alternate clicks

Triggering events

- Events can be programmatically triggered with .trigger() method
  - uses DOM APIs to force browser to trigger event
- Can also be triggered with .triggerHandler()
  - invokes handlers directly without using browser
  - does not bubble
- Shortcut methods that take in no arguments act as calls to .trigger()

```
$('#submit').click();
```

Custom events

- Any event type with an unrecognized name is a custom event
  - DOM will never trigger event, but you can with .trigger() or .triggerHandler()

```
$('#submit').on('foo', doFoo);
```

Removing event handlers

- Use .off() to remove event handlers
  - no arguments mean remove all event handlers for all event types
  - single eventType argument removes all event handlers of that type
  - eventType and handler arguments remove just that handler
  - does not work with anonymous functions!

```
$('#submit').off();                  // removes all handlers
$('#submit').off('click');           // removes click handlers
$('#submit').off('click', submit); // removes submit handler
```

Namespaced event handlers

- Event handlers can be placed in namespace using awkward convention
  - "eventType.namespace"
- Allows removal of all handlers in namespace
  - and also triggering of handlers in namespace

```
$('#submit').on('click.myEvents', validate)
            .on('click.myEvents', submit)
            .on('click', log);

$('#submit').off('.myEvents'); // removes first two only
```

“Live” events

- .on() can only bind handlers to existing elements
  - elements added to DOM after call to .on() require more calls to .on()
- .on() accepts optional selector to handle events for all current and future elements matching that selector
  - binds special handler to existing container elements
  - events bubble up and target tested against selector before invoking handler

```
$('#container').on('click', 'button', handleClick);
```

Summary

- jQuery makes cross-browser event handling easy and consistent
- Use custom events for application-specific, higher-level concepts
- Use event bubbling to ease the burden of handling events in dynamic documents