Front-end Masters
Day 2, 11/11/11

- Karl Swedberg
- kswedberg@gmail.com
- @kswedberg

Event Handling

Loading Events

- $(document).ready();
- .load()
  - Typically used with $(window)
  - Also helpful with images, but be careful!

Multiple “Ready” Handlers

- All are executed
- Execution takes place in order defined
- Can even be defined after the document is ready!

Low-level methods

.bind()

- add an event listener to elements
- can be a native event like "click"
- can be a custom event, triggered by other code

```
$('button').bind('click', function(event) {
  // button got clicked
});
```

Method Context

- The “this” keyword
- DOM Elements vs. jQuery Objects

```
$('button').bind('click', function(event) {
  // this is the <button> DOM element
  // this.className += ' clicked';
  // $(this) returns jQuery object w/ 1 element
  // $(this).addClass('clicked');
});
```

Low-level methods

.trigger()

- Trigger events programmatically
- Works for native and custom events

```
$('button').trigger('click');
```

Low-level methods

.bind() and .trigger()

```
// this listens for a click on a span
$('span').bind('click', function() {
  /* this stuff happens when a span is clicked */
});

// maybe the user will click on a span. Or...

// this triggers any click event bound to a span
$('span').trigger('click');
```

Low-level methods

.unbind()

- remove an event listener from elements
- remove all listeners, or just the specified one

```
$('button')
.bind('click', clickListener)
.bind('click', otherListener);

// unbind all click listeners
$('button').unbind('click');

// unbind only the specified listener
$('button').unbind('click', clickListener);
```

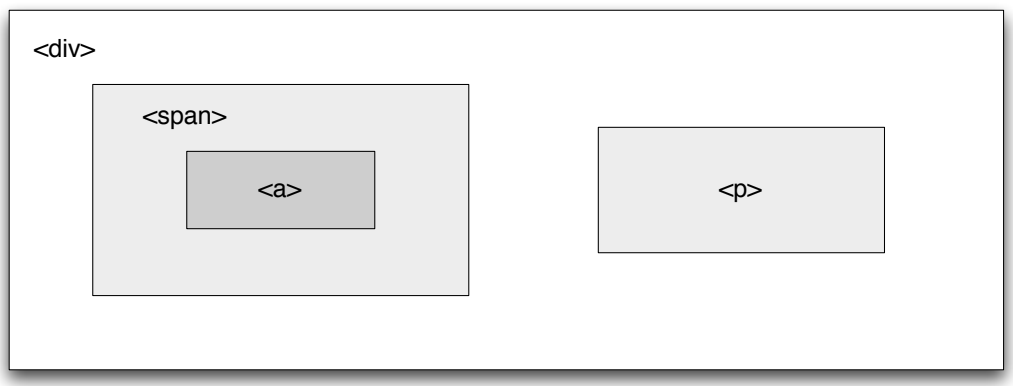
Shorthand Syntax

```
$('button').click(clickListener);

// == $('button').bind('click', clickListener);

$('button').click();

// == $('button').trigger('click');
```

Shorthand Syntax

- .submit()
- .change()
- .focus()
- .focusin()
- .blur()
- .focusout()
- .mouseover()
- .mouseout()
- .mouseenter()
- .mouseleave()
- .mousemove()
- .dblclick()
- .keydown()
- .keypress()
- .keyup()
- .scroll()
- .resize()
- .error()

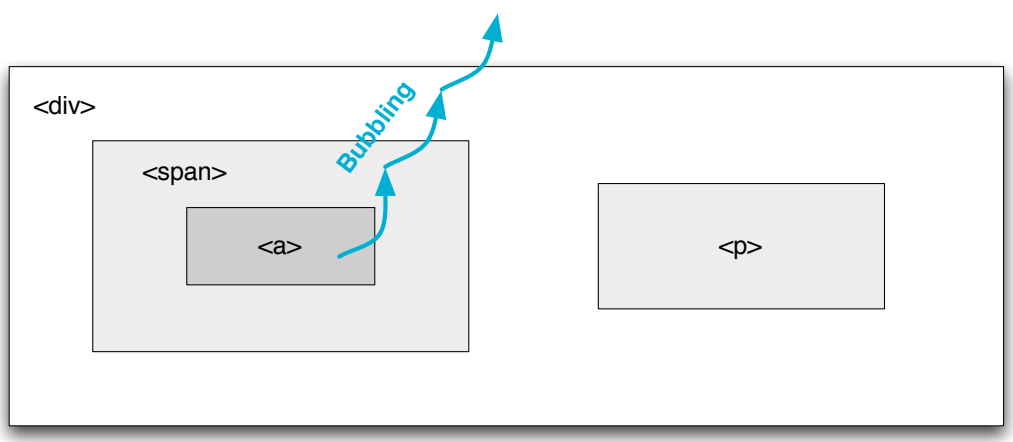
Event Propagation



a.k.a. "event bubbling"

http://www.quirksmode.org/js/events_order.html

Event Propagation



a.k.a. “event bubbling”

http://www.quirksmode.org/js/events_order.html

Two Event Tutorials

- How to ensure that new elements added to the DOM have events bound to them.
  - by using event delegation:
    Working with Events, part 1 (tinyurl.com/eventdelegation)
  - by re-binding:
    Working with Events, part 2 (tinyurl.com/eventrebinding)

Event Propagation

- Sometimes we don't want events to bubble.
  - Responding to hovers on menus
  - (which, maybe, we shouldn't do anyway)
  - mouseover vs. mouseenter demo

Event delegation

- alternative event handling architecture
  - scales much better
  - makes it trivial to add and remove elements
  - uses **event.target** rather than **this**
    *(but jQuery's delegation methods map *this* to *event.target*)
- don't ask every element for events, ask a parent instead
- effort is moved from binding to handling

Event Delegation

- .live()
  - binds an event handler to the document.
  - triggered when element acted upon or one of its ancestors matches selector
- .die()
  - unbinds an event handler

Event Delegation

- .live() and .die() look just like .bind() and .unbind()

```
$('button').live('click', function(event) {
  // button got clicked
});

// unbind click on button
$('button').die('click');
```

Event Delegation

- **Stop using .live() and .die()**
- They are *deprecated* as of jQuery 1.7
- They don't work as expected in some situations:

```
$('#wrapper').find('a, span').live('click', fn);
```

- They require events to bubble all the way up to `document`.
- **But**, if you're using them successfully with older jQuery versions, no worries!

Event Delegation

- .delegate()
  - binds an event handler to jQuery set.
  - triggered when element indicated in first argument is acted upon or one of its ancestors matches the first argument
  - more "optimized" delegation
- .undelegate()
  - unbinds an event handler

Event Delegation

```
$('#wrapper').delegate('button', 'click',
function(event) {
   // button got clicked
});

// unbind click on button
$('#wrapper').undelegate('button', 'click');
```

Event Delegation

- Use .delegate() and .undelegate() for event delegation in jQuery **before 1.7**.

Event Delegation

- .on()
  - new in jQuery 1.7
  - the future of jQuery event handling
  - one event handler method to rule them all
  - use instead of .bind() and .live() and .delegate()
- .off()
  - unbinds event handlers bound by .on()

Direct Event Binding

```
$('button')
.on('click', clickListener)
.on('click', otherListener);;

// remove all click listeners
$('button').off('click');

// remove only the specified listener
$('button').off('click', clickListener);
```

Event Delegation

```
$('#wrapper').on('click', 'button',
function(event) {
   // button got clicked
});

// remove click listeners
$('#wrapper').off('click', 'button');
```

event object

- normalized by jQuery to provide consistent information across browsers
- **event.target** to access the element that triggered the event
- **event.pageX/Y** for mouse position
- **event.preventDefault()** to prevent an event's default action
- **event.stopPropagation()** to prevent an event from bubbling
- **event.which** to identify which key is pressed
- more event properties at http://api.jquery.com/category/events/event-object/

event object example

- **prevent** the **default** action from occurring
- **stop** the event's **propagation** up the DOM

```
$('a.toggler').on('click', function(event) {
  event.preventDefault();
  event.stopPropagation();

  $(this).parent().next().slideToggle();

  // or, return false to both
  // preventDefault and stopPropagation
  // (but problematic)
});
```

event object example

- identify the **type** of event being triggered

```
$('button').on('mouseenter mouseleave', function(event) {
  var isEntered = event.type == 'mouseenter';

  $('#something').toggleClass('active-related', isEntered);

  if (isEntered) {
    // do one thing
  } else {
    // do another
  }
});
```

event object

- Very useful for key events
  - **event.which** : key code normalized by jQuery
  - **event.metaKey** : command / ⌘ on Mac, control on Win
  - **event.altKey** : alt / option / ⌥ on Mac
  - **event.ctrlKey** : control key everywhere
  - **event.shiftKey**
- key event demo

event object

- access newer event types through the **original event**

```
if (document.createTouch) {

  $('body').bind('touchstart touchmove touchend', function(event) {

    // use the original event:
    event = event.originalEvent;

   // pass the first touch object as a second argument
    var touch = event.targetTouches[0];
    sideSwipe[ event.type ](event, touch);
  });
}
```

what?

- `sideSwipe[ event.type ](event, touch);`

```
// declare some variables up here (startcoords, endcoords, etc.)

var sideSwipe = {
  touchstart: function(event, firstTouch) {
    startcoords = {x: firstTouch.pageX, y: firstTouch.pageY};
  },
  touchmove: function(event, firstTouch) {
    if (event.targetTouches.length === 1) {
      event.preventDefault();
      endcoords = {x: firstTouch.pageX, y: firstTouch.pageY};
    } else {
      endcoords = startcoords;
    }
  },
  touchend: function(event) {
    // direction of horizontal swipe?
    // also, far enough horiz, not too far vert.?
    // if so, do something
  }
};
```

what?

- `sideSwipe[ event.type ](event, touch);`

```
var sideSwipe = {
  touchstart: function(event, firstTouch) { },
  touchmove: function(event, firstTouch) { },
  touchend: function(event) { }
};

// if event.type equals 'touchstart', then:
/*
sideSwipe.touchstart() == sideSwipe['touchstart']()
sideSwipe['touchstart']() == sideSwipe[event.type]()
*/
```

Wait!

- There must be a better way...
- First argument for .on() and .bind() can accept a map of event types and their handler functions

```
var myEvents = {
  focus: function() {
    alert('focused!');
  },
  blur: function() {
    alert('blurry!');
  }
};

$('input').on( myEvents );
```

event map

```
var sideSwipe = {
  'touchstart touchmove': function(event) {
    event = event.originalEvent;
    var firstTouch = event.targetTouches[0];

    if (event.type == 'touchstart') {
      startcoords = {x: firstTouch.pageX, y: firstTouch.pageY};
      return;
    }

    if (event.targetTouches.length === 1) {
      event.preventDefault();
      endcoords = {x: firstTouch.pageX, y: firstTouch.pageY};
    } else {
      endcoords = startcoords;
    }
  },
  touchend: function(event) {
    // get direction of horizontal swipe
    // also, far enough horiz, not too far vert.?
    // if so, do something
  }
};
```

event map

```
if (document.createTouch) {

  $(document.body).on(sideSwipe);

}
```

Namespacing events

- Namespace events to ease unbinding
- Especially nice in plugins

```
var $foo = $('#foo');
var karlClicks = function() { /*do something*/ };

$foo.on('mouseenter.karl', function() { /*do something*/ });
$foo.on('click.karl', karlClicks);

$foo.on('click', function() { /* do something */ });



$foo.off('.karl'); // stop doing .karl things
```

Custom Events

- Awesome for "evented" programming
- Set up a bunch of behaviors all in one place.
- Fire at will (**.trigger()**)

Custom Events Fake Example

```
$(document).on('start.game', myGameStartFn);
$(document).on('stop.game', myGameStopFn);

$(document).on('updateScore.game', function(event, data) {
  $.ajax('/gameserver/', data);
  $('#scoreboard .score').html( data.score );
});

$(document).on('updateBoard.game', function(event, data) {
  if (data.killed) {
    $(event.target).fadeOut(400, function() {
      $(this).remove();
    });
  }
});

$('div.alien').on('click', function(event) {
  score += 1000;
  var data = {
    score: score,
    killed: this.className,
    name: $(this).data('name')
  };

  $(this)
  .trigger('updateBoard.game')
  .trigger('updateScore.game', [data]);
  if ( score > 1000000) {
    $(document).trigger('stop.game');
  }
});
```

DOM Manipulation

DOM Manipulation

- Create Elements
- Insert Elements
  - Element Content
- Remove Elements
- Element Properties and Attributes
- Styles and Dimensions

```
$('div')
```

```
$('<div></div>')
```

```
$('<div/>')
```


```
$('<div/>', {
  'class': 'test',
  html: '<b>Click me!</b>',
  click: function(){
    $(this).toggleClass('test');
  }
});
```

```
$('<div/>', {
  'class': 'test',
  html: '<b>Click me!</b>',
  click: function(){
    $(this).toggleClass('test');
  }
}).appendTo('body');
```

Clone Elements

- Clone elements via $('#myid')**.clone()**
  - .clone(**true**) to copy events and data, too

Insert Elements

(or move them)

- add and move elements
  - inside others with **.append()**, **.appendTo()**, **.prepend()** and **.prependTo()**
  - before and after others with **.after()**, **.insertAfter()**, **.before()** and **.insertBefore()**
  - around others with **.wrap()**, **.wrapAll()** and **.wrapInner()**
  - in place of others with **.replaceWith()** and **.replaceAll()**

Insert Elements

(or move them)

```
$('<div></div>').appendTo('#mydiv');
$('#mydiv').append('<div></div>');

$('<div></div>').insertAfter('#mydiv');
$('#mydiv').after('<div></div>');

// and so on
```

Insert Elements

(or move them)

- In 1.4 and above, a subset of these methods can take a function
  - **.append()**
  - **.prepend()**
  - **.after()**
  - **.before()**
  - **.wrap()**
  - **.wrapInner()**
- In the function, you'll need to *return* the content you want to insert

Remove Elements

- Remove elements with **.remove()**
  - Cleans up data and events
- Remove elements with **.detach()**
  - Leaves data and events alone
- Remove their content with **.empty()**
  - Cleans up data and events

Element Content

- Get and set HTML content via **.html()**
  - executes embedded JavaScript when setting
- Get and set text content via **.text()**
  - escapes HTML entities (<, >, &)
  - recursive (drills down / concatenates)
- Get and set form element values with **.val()**

Attributes & Properties

- Get and set any property with **.prop()**
- Remove properties with **.removeProp()**
- Get and set any attribute with **.attr()**
- Remove attributes with **.removeAttr()**

Attributes & Properties

what's the difference?

- Attributes are what you see in the HTML.
- Properties are ... well ... *properties* of DOM elements (objects)

Attributes & Properties

what's the difference?

- Some props don't have attr equivalents
- Some props and attrs have different names

prop	attr
selectedIndex, tagName, nodeName, nodeType, ownerDocument	nothing, nada, zilch, nil, goose egg
htmlFor	for
className	class

Attributes & Properties

what's the difference?

- Some attributes are considered *boolean* attributes
- For these, you really ought to get and set with *.prop()*
- *... even though jQuery will still coddle you if you do it wrong*

Attributes & Properties

what's the difference?

```
<input type="checkbox" checked="checked">

<script>
// DOM property

// Will change with checkbox state
elem.checked == true;

// Will change with checkbox state
$(elem).prop("checked") == true;

</script>
```

Attributes & Properties

what's the difference?

```
<input type="checkbox" checked="checked">

<script>
// DOM method

// Initial state of the checkbox; does not change
elem.getAttribute("checked") == "checked";

//(1.6) Initial state of the checkbox; does not change
$(elem).attr("checked") == "checked";

//(1.6.1+) Will change with checkbox state
$(elem).attr("checked") == "checked";

//(pre-1.6) Changed with checkbox state
$(elem).attr("checked") == true;
</script>
```

Attributes & Properties

- See why you should use .prop() ?
- (applies to "selected", too)

Attributes & Properties

- Remember me?

```
$('<div/>', {
  'class': 'test'
});
```

- I use **.attr()** (so you must use "class", not "className")

Attributes

- get attribute:

```
$('a').attr('href');
```

Attributes

- set attribute:

```
$('input').attr({
  title: 'this is my title',
  name: 'some_name'
});

$('input').attr('title', 'this is my title');
```

Attributes

- set attribute:

```
$('img').attr({
  title: function(index, value) {
    return 'I do believe that ' + value;
  },
  name: 'some_name'
});

$('a').attr('title', function() {
  return 'go to ' + this.href;
});
```

Properties

- get prop:

```
$('a').prop('id');
$('a').prop('href'); // this one is problematic!
```

Properties

- set property:

```
$('input').prop({
  title: 'this is my title',
  name: 'some_name'
});

$('input').prop('title', 'this is my title');
```

Properties

- set property:

```
$('img').prop({
  title: function(index, value) {
    return 'I do believe that ' + value;
  },
  name: 'some_name'
});

$('a').prop('title', function() {
  return 'go to ' + this.href;
});
```

Styles

- Use **.css()** to modify the style property
- Use **.addClass()**, **.removeClass()** and **.toggleClass()** to modify the class attribute
  - When possible, manipulate the class attribute to change appearance, separating the actual design from the behavior

Style Properties

- get style property
  - `$('a').css('color');`

Style Properties

- set style property

```
$('a').css({
  backgroundColor: 'red',
  'margin-top': '10px'
});

$('a').css('backgroundColor', 'red');
```

either DOM prop or CSS syntax

Notice!

jQuery
write less, do more.

Style Properties

- set style property:

```
$('a').css('fontSize', function() {
  return parseInt($(this).css('fontSize'),10) + 2 + 'px';
});

$('a').css({
  backgroundColor: 'red',
  marginTop: function(index, value) {
    return parseInt(value,10) * 1.4;
  }
});
```

Dimensions

- Get and set width and height via **.width()** and **.height()**
  - return a number
  - nice for calculations
- Get and set the element offset relative to the document with **.offset()**
  - returns an object
    - e.g. {top: 34, left: 50}
- Get the element offset relative to its "offset parent" with **.position()**

Data

- associate data with elements
  - plugin settings
  - plugin instances
- avoid storing data directly on DOM elements
  - avoid memory leaks
- Set data: **$('div').data('personalInfo', {firstname:'karl'});**
- Get data: **$('div').data('personalInfo');**
- Remove data: **$('div').removeData('personalInfo');**

Data

- As of jQuery 1.4.4, can read HTML5 data-* attributes.
- jQuery guesses what the type is and treats it accordingly.
- To parse a data value as an *object*, it must appear as valid JSON.
- Read learningjquery.com/2011/09/using-jquerys-data-apis

- <div **data-img**='{"alt":"pic","src":"path/file.jpg"}'>

- $('div').data(**'img'**);

Q & A

and look at examples

Ajax

Ajax

- Unobtrusive client-server data exchange
  - avoid page refresh
- Affords much more interactive applications
  - becoming nearly indistinguishable from desktop applications
- Progressive Enhancement is essential

$.ajax

- jQuery's low-level abstraction for Ajax
- all high-level methods build on it
- Configured per request or global defaults

$.ajax options: url

- Address of the server-side resource
- Can contain query string
  - use *data* instead

```
$.ajax({
   url: '/url/to/serverResource'
});
```

$.ajax options: url

- Address of the server-side resource
- Can be passed as a string to first argument (as of 1.5)

```
$.ajax('/url/to/serverResource');
```

$.ajax options: data

- To send information from client to server
- with GET: Appended to url as query string
- with POST: Sent as POST body

```
$.ajax({
   url: '/url/to/serverResource',
   data: {
      key1: 'value1',
      key2: 'value2'
   }
});
```

$.ajax options: data

- When submitting a form, use **.serialize( )**

```
$('#myform').submit(function(event) {
  event.preventDefault();

  var formUrl = $(this).attr('action'),
      formData = $(this).serialize();

  $.ajax({
    url: formUrl,
    data: formData
  });

});
```

$.ajax options: dataType

- '**html**' for HTML to insert into document
- '**xml**' for XML documents, eg. web services
- '**json**' for compact alternative to XML
  - parsed using browser's native JSON parser if available
  - much easier to use in JS than XML
- '**jsonp**' for remote resources. Gets around same-origin policy.
  - Response acts like a function with a JSON string as its argument. jQuery calls it and parses the JSON.
- '**script**' for JavaScript files
  - Evaluates the response as JavaScript and returns it as plain text.

$.ajax options: dataType

- Specifies expected response
- Default based on MIME Type of the response

```
$.ajax({
   url: '/url/to/serverResource',
   dataType: 'json'
});
```

$.ajax options: Lots More

- Read all about 'em at api.jquery.com/jQuery.ajax/

$.ajaxSetup( )

- Global ajax settings

$.ajax responses

- Before jQuery 1.5, these were handled by three more options (!!)
  - { **success**: function(){}, **error**: function(){}, **complete**: function(){} }
- don't use them anymore

```
$.ajax({
  url: '/url/to/serverResource',
  success: function(response, status, xhr) {
    // do something after successful request
  },
  error: function(xhr, status, errorThrown) {
    // handle failed request
  },
  complete: function(xhr, status) {
    // do something whether success or error
  }
});
```

Ajax error handling

- Explicit testing for ajax errors is important
  - Network failures don't occur in local development environment
- p.s. JSON parsing errors throw an exception

$.ajax responses

- $.ajax implements the Promise interface
  - returns a jqXHR object (superset of xhr), a Promise
  - Promise objects are derived from the Deferred object
  - Not enough time to dive into that today
  - Read api.jquery.com/category/deferred-object/
- jQuery 1.5+:
  - .**done**() and .**fail**() methods
- jQuery 1.6+:
  - .**always**() method

$.ajax responses

1.5+

- Methods can be called multiple times to add more than one handler.
- Can store result of Ajax request in variable and attach handlers later for more manageable code structure.
- Handlers will be invoked immediately if the Ajax operation is already complete when they are attached
- Ajax requests can be cached in a simple, elegant way

$.ajax responses

1.5+

```
  var myOptions = {
  url: 'http://api.jquery.com/jsonp/',
  dataType: 'jsonp',
  data: {
    title: search
  }
};

$.ajax( myOptions )
.done( successFn )
.fail( errorFn )
.always( completeFn );
```

$.ajax responses

1.5+

- Multiple function arguments
- Array of functions

```
request.done(successFnA, successFnB, successFnC);

request.done([successFnD, successFnE, successFnF]);

request.done(successFnG).done(successFnH).done(successFnJ);
```

Caching Ajax Responses

- A simple approach
- For more generic, abstracted approach, see Script Junkie: "Creating Responsive Applications Using jQuery Deferred and Promises" bit.ly/tph6F6

Caching Ajax Responses

```
  (function() {
var api = {}, $response = $('#response');

$('#ajaxForm').bind('submit', function(event) {
  event.preventDefault();
  var search = $('#title').val();
  $response.empty().addClass('loading');









  });
})();
```

Caching Ajax Responses

```
(function() {
  var api = {}, $response = $('#response');

  $('#ajaxForm').bind('submit', function(event) {
    event.preventDefault();
    var search = $('#title').val();
    $response.empty().addClass('loading');


    var ajaxResults = $.ajax({
      url: 'http://api.jquery.com/jsonp/',
      dataType: 'jsonp',
      data: {
        title: search
      },
      timeout: 15000
    });

    ajaxResults.done(successFn).fail(errorFn).always(completeFn);
  });
})();
```

Caching Ajax Responses

```
(function() {
  var api = {}, $response = $('#response');

  $('#ajaxForm').bind('submit', function(event) {
    event.preventDefault();
    var search = $('#title').val();
    $response.empty().addClass('loading');


    api[search] = $.ajax({
      url: 'http://api.jquery.com/jsonp/',
      dataType: 'jsonp',
      data: {
        title: search
      },
      timeout: 15000
    });

    api[search].done(successFn).fail(errorFn).always(completeFn);
  });
})();
```

Caching Ajax Responses

```
(function() {
  var api = {}, $response = $('#response');

  $('#ajaxForm').bind('submit', function(event) {
    event.preventDefault();
    var search = $('#title').val();
    $response.empty().addClass('loading');

    if (!api[search]) {
      api[search] = $.ajax({
        url: 'http://api.jquery.com/jsonp/',
        dataType: 'jsonp',
        data: {
          title: search
        },
        timeout: 15000
      });
    }
    api[search].done(successFn).fail(errorFn).always(completeFn);
  });
})();
```

Caching Ajax Responses

```
(function() {
  var api = {}, $response = $('#response');

  $('#ajaxForm').bind('submit', function(event) {
    event.preventDefault();
    var search = $('#title').val();
    $response.empty().addClass('loading');

    if (!api[search]) {
      api[search] = $.ajax({
        url: 'http://api.jquery.com/jsonp/',
        dataType: 'jsonp',
        data: {
          title: search
        },
        timeout: 15000
      });
    }
    api[search].done(successFn).fail(errorFn).always(completeFn);
  });
})();
```

Ajax Convenience Methods

- $.get()
- $.post()
- $.getJSON()
- $.getScript()

Ajax Convenience Methods

- $('elem').**load( )**
- The "convenientest" of all …
- Single argument can do it all. Often no need for callback function.

```
$('#myid').load('/foo.html #anotherid');

$('#myid').load('/foo.html #anotherid', function() {
// do something on complete
});
```

Ajax Convenience Methods

- My recommendation:
- **Use $.ajax()**

Effects

Slides
Fades
Custom animations

Introduction

- Effects can enhance user interaction
- Prefer subtle over in-your-face
- jQuery provides a base set of animations
- make animations:
  - long enough to be noticeable
  - short enough to not annoy

Fades

- great to show/hide overlay elements
  - tooltips
  - dialogs
  - warning/info messages
- Examples:
  - `$('.tooltip').fadeIn();`
  - `$('.tooltip').fadeOut('slow');`
  - `$('.tooltip').fadeToggle(250);`
  - `$('.tooltip').fadeTo(200, 0.5);`

Slides

- show and hide elements within the page structure
- less jarring than un-animated show/hide

```
$('div.panel')
.wrapAll('<div class="panel-wrapper"></div>');

$('div.panel-heading a').click(function() {
  $(this).parent().next('.panel-wrapper')
  .slideToggle();
  return false;
});
```

Callback

- Function executed when the animation ends
- Called once for *each* animated element
- Consider using .promise()

```
$('div.toRemove').slideUp('slow', function() {
  $(this).remove();
});

$('div.move').slideDown(250).promise().done(function() {
  alert('Finished!');
});
```

Animation Order

- By default multiple animations occur:
  - in sequence for the same element(s)
  - simultaneously for different element(s)

Custom Animations

- $('div.toMove').**animate**(propsObject, duration, easing, callbackFn);
- $('div.toMove').**animate**(propsObject, optionsObject);

- Properties can be animated
  - by number, percent, etc.
  - relatively ("+=200px", "-=20%", etc.)
  - by keyword: "hide", "show", or "toggle"

Custom Animations

- Create custom animations
- Example, slowly moving an element 300px to the right:

```
$('.toMove').animate({
  left: '+=300px'
}, 800);

// same thing
$('.toMove').animate({
  left: '+=300px'
}, {
  duration: 800
});
```

Custom Animations

- Options object allows for fine tuning:
  - **duration**: A string or number determining how long the animation will run.
  - **easing**: A string indicating which easing function to use for the transition. ("linear", "swing". More with plugin.)
  - **complete**: A function to call once the animation is complete. (callback function.)
  - **step**: A function to be called after each step of the animation.
  - **queue**: A Boolean indicating whether to place the animation in the effects queue. If false, the animation will begin immediately.
  - **specialEasing**: A map of one or more of the CSS properties defined by the properties argument and their corresponding easing functions (added 1.4).

Easing

- Changes the velocity at different points in the animation
- A number of standard equations first created by Robert Penner
  - Available with jQuery UI
  - Also, stand-alone at http://gsgd.co.uk/sandbox/jquery/easing/

```
$('#foo').animate({height: 'toggle'},  {
  duration: 600,
  easing: 'linear'
});
```

Easing

- Per-property easing available as of jQuery 1.4.

```
$('#clickme').click(function() {
    $('#book').animate({
      width: ['toggle', 'swing'],
      height: ['toggle', 'swing'],
      opacity: 'toggle'
    }, 5000, 'linear', function() {
      $(this).after('<div>complete!</div>');
    });
});
```

Currently Animated

- Identify elements that are currently animated

```
$('.toMove').click(function() {
  if (!$(this).is(':animated')) {
    $(this).animate({
      left: '+=300px'
    }, 'slow');
  }
});
```

Stop

- Stop current animations from continuing.
- Two arguments: clearQueue and gotoEnd (both boolean)

```
$('#badges li').hover(function(){
  $(this).stop(true, true)
  .animate({bottom:"30px"}, 200);
}, function(){
  $(this).stop(true, false)
  .animate({bottom:"8px"}, 500);
});
```

A Quick Example

LittleOrangeStar.com

Delay

- Delays any further items in the queue from executing for a specified period of time.

```
$('#warning').fadeIn(600).delay(4000).fadeOut();
```

Global Duration

- Globally change the duration that all animations will use unless a duration is specified.

```
// animation will complete in 400ms
$('#foo').slideToggle();

// modify default globally for all future effects
$.fx.speeds._default = 250;

// animation will complete in 250ms
$('#foo').slideToggle();
```

Global Off

- Globally prevent all animations from occurring

```
$('.animations').click(function() {
  $.fx.off = !$.fx.off;
});
```

Thank You!