jQuery and Ajax

(not for the dummies)

1. Introduction

jQuery (@ http://jquery.com) is a JavaScript Library. It is a small script (about 96kB minified) written in JavaScript called "`jquery.js`", which greatly simplifies JavaScript programming by providing cross-browser supports for DOM element selection and manipulation, event handling, Ajax request/response processing and animation.

jQuery is highly popular. In May 2015, JQuery is used by 64.2% of all the websites. Among the JavaScript libraries/frameworks, jQuery's market share is 95.2% (Ref: http://w3techs.com/technologies/overview/javascript_library/all). In other words, most of the developers nowadays program in jQuery, rather than raw JavaScript.

jQuery vs. Raw JavaScript

- **Cross-Browser Support**: jQuery provides cross-browser support. That is, the same jQuery code runs on the big-5 browsers (Chrome, Firefox, IE, Safari and Opera). On the other hand, to provide cross-browser support in raw JavaScript, you need to check the browser and issue the appropriate codes, as different browsers (particularly IE) implement certain features differently. This is done implicitly in jQuery.
- **DOM Elements Selection and Manipulation**: The "query" refers to querying and selecting DOM elements within a web document for subsequent manipulation. jQuery provides a powerful and supercharged selector function to select elements based on HTML tag-names (e.g., `<p>`, `<button>`), HTML ID attribute (e.g., `#debug`), and CSS class name (e.g., `.error`). On the other hand, selecting and manipulating DOM elements using raw JavaScript is messy and cumbersome.
- **Event Handling**: jQuery also simplifies JavaScript event handling.
- **Special Effects and Animation**: jQuery simplifies the programming for special visual effects (such as show/hide, fade-in/fade-out, slide-in/Slide-out) and custom animation.
- **AJAX Interface**: jQuery provides a simple Ajax interface to send asynchronous HTTP GET/POST requests and process the response.

With jQuery, you can write a few lines of codes to replace tenths of JavaScript codes; and run on all browsers without the need to test on each of them. The cross-browser support is particularly important for production, as you can't possibly test your JavaScript codes on all browsers. jQuery is well-tried. It is reported that jQuery is used by over 60% of the production websites in the Internet!

I shall assume that you are familiar with HTML5, CSS3 and JavaScript, which are the absolutely necessary pre-requisites for using jQuery. Remember that jQuery is written in JavaScript!

2. Using jQuery

Installation and Setup

1. Download jQuery library from http://jquery.com.
2. Copy the JavaScript file (e.g., `jquery-1.xx.x.min.js`) under your document root directory, typically under a sub-directory "`js`".
   Note: The "`min.js`" is the *minified* version meant for production, which removes additional spaces and comments to reduce the file size for faster download. For testing and studying the codes, use the "`.js`" version.
3. Include in your HTML document:

   ```
   <script src="js/jquery-1.xx.x.min.js"></script>
   ```

   This is typically place in the `<head>` section. But you can place it anywhere in the document, as long as before any jQuery function (such as `$()`) is used.
   Note: In HTML4/XHTML1.0, you need to include attribute type=`"text/javascript"` in the `<script>` opening tag.

Using the jQuery CDN

Alternatively, instead of serving the `jquery.js` from your server, you can use one of the CDN (Content Distribution Network) to serve it. This could save some of your network traffic and probably provide slightly faster response. Moreover, the download `jquery.js` would be cached for reuse.

- jQuery.com's CDN

  ```
  <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
  ```

- Google CDN

  ```
  <script
  src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></
  script>
  ```

- Microsoft CDN

  ```
  <script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-
  1.11.3.min.js"></script>
  ```

jQuery Versions

jQuery has two versions. jQuery version 2 does not support IE <9 versions. As there are still quite a number of old IEs (v7, v8) around, jQuery version 1 is a lot more popular in production systems.

jQuery Template

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="utf-8">
5  <title>YOUR TITLE HERE</title>
6  <script src="js/jquery-1.11.2.min.js"></script>
7  <script>
8  // Run after the Document DOM tree is constructed
9  $(document).ready( function() {  // or shorthand of $( function () {
```

```
  // Your jQuery scripts here!
});
</script>
</head>
<body>
<h1>Hello, world!</h1>
</body>
</html>
```

Notes:

1. Some people prefers to place the JavaScripts just before the end of body (`</body>`), instead of `<head>` section, for slightly better responsiveness.
2. Load the CSS before the JavaScripts, as JavaScripts often reference the CSS.

3. jQuery By Examples

3.1 Example 1: jQuery Selectors and Operations

"**JQEx1.html**"

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Example 1: jQuery Selector and Operations</title>
<script src="js/jquery-1.11.2.min.js"></script>
<script>
// Run after the ROM tree is constructed.
$(document).ready( function() {
   // Select an element that matches the element's unique id
   $('#hello').html('Hello, world!');   // Replace innerHTML
   $('#hello').addClass('green');       // Add CSS class

   // Select an element that matches the element's "unique id"
   $('#message').append("(id matched)");  // Append at the end

   // Select element(s) that match "HTML tag name" and process via implicit loop
   $('p').prepend("(tag-name matched)");  // Add in front

   // Select element(s) that match the "CSS classname" and process via explicit loop
   $('.red').each( function() {
      $(this).append("(class-name matched)");
      $(this).prepend("(class-name matched)");
   });

   // Apply many operations via chaining
   $('.red').before("<p>Before</p>")   // before the current element
            .after("<p>After</p>");    // after the current element
});
</script>
<style>
.red   { color: #FF0000; }
.green { color: #00FF00; }
.blue  { color: #0000FF; }
</style>
</head>
<body>
  <h1 id="hello">Hi!</h1>
  <p id="message" class="red">First Line </p>
  <p class="red">Second Line </p>
```

```
41   <p>Third Line </p>
42 </body>
43 </html>
```

How it Works?

1. The "query" in jQuery refers to *querying* or *selecting* element(s) in an HTML document for subsequent manipulations. For examples,
    - `$(document)` selects the current document;
    - `$(p)` selects all the `<p>` elements (Tag-Selector);
    - `$(#hello)` and `$(#message)` select one element having attribute `id="hello"` (ID-Selector);
    - `$(.red)` selects all the elements having attribute `class="red"` (Class-Selector).
    - In fact, `$()` is the shorthand (alias) for the main `jQuery()` function.
2. jQuery selector - the most important jQuery function - has a special syntax of `$()`. It could take a tag name, an id attribute (with prefix of #) or classname (with prefix of dot). In fact, it supports all the CSS Selectors!
3. Comparing with JavaScript's many selector functions (such as `document.getElementById()`, `document.getElementsByTagName()`, `document.getElementsByClassName()`, `document.getElementsByName()`, `document.querySelector()`, `document.querySelectorAll()`), jQuery's selector is much simple and one class above.
   The `$(document).ready(`*`handler`*`)` attaches an event handler, which will be fired once the DOM tree is constructed. The "`ready`" event (new in jQuery) is slightly different from the JavaScript's "`onload`" event, which does not wait for the external references (such as images) to be loaded. We wrap our jQuery operations under the `ready()`, as these codes are placed in the `<head>` section, before the referenced elements are constructed in the `<body>`. This is a common jQuery practice.
4. There are various methods available for manipulating the contents of the selected element(s). For example,
    - `html()`: get the innerHTML.
    - `html(`*`value`*`)`: set the innerHTML.
    - `append(`*`value`*`)`: append at the end of the innerHTML.
    - `prepend(`*`value`*`)`: add in front of the innerHTML.
    - `before(`*`element`*`)`: add the *`element`* before the current element.
    - `after(`*`element`*`)`: add the *`element`* after the current element.
    - `addClass(`*`value`*`), removeClass(`*`value`*`), toggleClass(`*`value`*`)`: add, remove or toggle a value of the `class` attribute.
5. jQuery builds in an *automatic looping* feature (Line ? to ?). For example, `$('p')` selects all `<p>` elements. `$('p').append(...)` applies the `append(...)` to each of the selected `<p>` element, in a implicit loop.
6. You can also use an *explicit loop* via `.each( function() {...} )` (Line ? to ?), if you need to apply more than one operations to the selected elements. Inside the `.each(...)`, the `$(this)` denotes the element under operation.

7. You can also use *function chaining* to chain the functions (Line ? to ?), as most of the functions return the element under operation.
8. In many methods (such as `html()`), jQuery uses the same method name for both *getter* and *setter*, differentiated by its argument. For example `html()` (without argument) returns the innerHTML, while `html(`*`value`*`)` replaces the innerHTML. [It does not use Java's convention of `getHtml()` and `setHtml()`.]
9. The `$(document).ready(function() {...})` runs the functions after the ROM tree is constructed, but does not wait for all the external resources (such as images) to be loaded (as in the JavaScript `load` event). Document `ready` is commonly used in jQuery, which provides a shorthand written as `$(function() {...})`.

3.2 Example 2: jQuery Event Handling

"JQEx2.html"

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="utf-8">
5  <title>jQuery Example 2: Event Handling</title>
6  <script src="js/jquery-1.11.2.min.js"></script>
7  <script>
8  // Run after the ROM tree is constructed.
9  $(document).ready( function() {
10    // Set the content
11    $('#hello').html('Click me!');
12
13    // Bind a onclick handler to a selected element
14    $('#hello').click(function() {
15       $(this).html('Hello, world!');
16       return false;   // Prevent triggering the default handler
17    });
18
19    // Bind onmouseover/onmouseout handlers to all selected elements
20    $('p').mouseover(function() {
21       $(this).addClass('green');
22    });
23    $('p').mouseout(function() {
24       $(this).removeClass('green');
25    });
26 });
27 </script>
28 <style>
29 .red   { color: #FF0000; }
30 .green { color: #00FF00; }
31 .blue  { color: #0000FF; }
32 </style>
33 </head>
34 <body>
35   <h1 id="hello"> </h1>
36   <p id="message" class="red">First Line </p>
37   <p class="red">Second Line </p>
38   <p>Third Line </p>
39 </body>
40 </html>
```

How it Works?

1. Example 1 illustrate the jQuery selector and built-in functions. But Example 1 is a useless as all changes are pre-program, instead of responding to the user's action. This example shows you how to program event handler to handle user's action. Most of the jQuery codes is actually dealing with programming event handlers for a set of selected elements! The steps are:
    1. Select the source elements via an appropriate jQuery selector.
    2. Identify the event, such as mouse-click, key-type.
    3. Write the event handler, and attach to the source.
2. You could attach an event handler to a JavaScript event, such as `click`, `mouseover` and `submit`, to the selected element(s) via jQuery methods, such as `.click(`*`handler`*`)`, `.mouseover(`*`handler`*`)`, `.submit(`*`handler`*`)`, as shown. You can prevent the default handler from being triggered by returning `false` from your event handler.
3. Inside the function, `$(this)` refers to the current object. Although `$(p)` returns more than one elements (in an array), you can use the same syntax to bind an event handler to EACH of the elements.
4. In the past, we placed the JavaScript event handler inside the HTML tags, e.g., "`<h1 onclick='....'>`". The practice nowadays is to leave them outside the HTML tags, and group them under the `<script>` section, for better MVC design.

3.3 Example 3: AJAX Request/Response

To test AJAX, you need to run the script under a web server (such as Apache).

"JQEx3.html"

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Example 3: Ajax Request/Response</title>
<script src="js/jquery-1.11.2.min.js"></script>
<script>
$(document).ready(function() {
   // Bind submit button onclick handler to send an Ajax request and
   //  process Ajax response.
   $(':submit').click(function (event) {
      event.preventDefault();  // Do not run the default action
      var submittedMessage = $(':text[name="message"]').val();
      $.ajax({
         type: 'POST',
         url:  'ProcessMessage.php',
         data: { message: submittedMessage }
      })
         .done( function (responseText) {
            // Triggered if response status code is 200 (OK)
            $('#message').html('Your message is: ' + responseText);
         })
         .fail( function (jqXHR, status, error) {
            // Triggered if response status code is NOT 200 (OK)
            alert(jqXHR.responseText);
         })
         .always( function() {
            // Always run after .done() or .fail()
            $('p:first').after('<p>Thank you.</p>');
         });
```

```
31    });
32 });
33 </script>
34 </head>
35 <body>
36 <form method="POST">
37   <label>Enter your message: <input type="text" name="message"></label><br>
38   <input type="submit">
39 </form>
40 <p id="message"> </p>
41 </body>
42 </html>
```

"**ProcessMessage.php**"

```
<?php
// Echo the POST parameter "message"
echo $_POST['message'];
?>
```

How it Works?

1. The `$(:submit)` selector selects all `<input type="submit">` elements.
2. The `$(:text[name="message"])` select `<input type="text" name="message">` elements. The `.val()` returns the `value` of the input text element.
3. We can use `$.ajax()` to send an Ajax request:
    - `.ajax()` takes an associative array (of key-value pairs) as its argument. The key `type` specifies the request method (such as `get` or `post`). The key `url` specifies the action url, default to current document. The key `data` provides the query string, in the form of an associative array of `{paramName:paramValue}` (as in the above example); or a proper query string (e.g., `name=peter&message=Hello`). Use the qurey string format if your parameter has multiple values (e.g., `name=peter&message[]=Hello&message[]=Hi`).
    - The `.done()` is called back when the response is received with status code of 200 (OK). It take a function with the HTTP response message as argument.
    - The `.fail()` is called back when the response is received with status code of NOT 200 (OK). It take a function with 3 arguments: `xhr` (XMLHttpRequest), `status` and `error`. You can get the response text via property `xhr.responseText`.
    - The `.always()` is called back after the `.done` or `.fail` completes. It takes a no-arg function as its argument.
    - Note that `.done()`, `.fail()` and `.always()` are chained together. *Function Chaining* is used extensively in jQuery.
4. The `$('p:first')` selects the first `<p>` element in the document. The `.after(`*`element`*`)` inserts the element after the current element.
5. The "`ProcessingMessage.php`" simply returns the value of POST parameter "`message`".

3.4 Example 4: Animation and Special Visual Effects

jQuery makes applying special effects and animation simple.

"**JQEx4.html**"

```
<!DOCTYPE html>
<!-- jQuery Example 4: JQEx4.html -->
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Example 4: Animation and Special Effects</title>
<script src="js/jquery-1.11.2.min.js"></script>
<script>
// Run after the Document DOM tree is constructed
// to bind click event handle to all the buttons
$( function() {
   // Show it by popping up
   $(':button[id="btnShow"]').click( function() {
      $('#logo').show();
   });

   // Hide it by popping out
   $(':button[id="btnHide"]').click( function() {
      $('#logo').hide();
   });

   // If it is visible, hide it; otherwise, show it
   $(':button[id="btnToggle"]').click( function() {
      $('#logo').toggle();
   });

   // Show by fading in
   $(':button[id="btnFadeIn"]').click( function() {
      $('#logo').fadeIn(1000);  // Speed: 1 sec
   });

   // Hide by fading out
   $(':button[id="btnFadeOut"]').click( function() {
      $('#logo').fadeOut(
         2000,                // Speed: 2 sec
         function() {         // Callback when complete
            alert('done!');
         });
   });

   // If it is visible, fade-out; Otherwise, fade-in
   $(':button[id="btnFadeToggle"]').click( function() {
      $('#logo').fadeToggle(3000);  // Speed: 3 sec
   });

   // Hide by sliding-up to top-left corner
   $(':button[id="btnSlideUp"]').click( function() {
      $('#logo').slideUp();   // disappear,
   });

   // Show by sliding-down from top-left corner
   $(':button[id="btnSlideDown"]').click( function() {
      $('#logo').slideDown();
   });

   // If it is visible, slide-up; Otherwise, slide-down
   $(':button[id="btnSlideToggle"]').click( function() {
      $('#logo').slideToggle();
   });


   // Custom animation, by applying given CSS properties
```

```
63      var toggleFlag = true;
64      $(':button[id="btnAnimate"]').click( function() {
65         if (toggleFlag) {
66            $('#logo')
67               .show()
68               .animate(
69                  { 'margin-left': '30px', // Apply these CSS properties
70                    'margin-top' : '20px',
71                    'opacity': 0.2      // semi-transparent
72                  },
73                  2000  // Speed: 2 sec
74               );
75         } else {
76            $('#logo')
77               .show()
78               .animate(
79                  { 'margin-left': '0px', // Apply these CSS properties
80                    'margin-top' : '0px',
81                    'opacity': 1.0        // not transparent
82                  },
83                  3000  // Speed: 3 sec
84               );
85         }
86         toggleFlag = !toggleFlag;
87      });
88  });
89  </script>
90
91  </head>
92
93  <body>
94  <p>Hello, world!</p>
95  <input type="button" id="btnShow" value="Show">
96  <input type="button" id="btnHide" value="Hide">
97  <input type="button" id="btnToggle" value="Toggle">
98  <input type="button" id="btnFadeIn" value="Fade-In">
99  <input type="button" id="btnFadeOut" value="Fade-Out">
100 <input type="button" id="btnFadeToggle" value="Fade-Toggle">
101 <input type="button" id="btnSlideUp" value="Slide-Up">
102 <input type="button" id="btnSlideDown" value="Slide-Down">
103 <input type="button" id="btnSlideToggle" value="Slide-Toggle">
104 <input type="button" id="btnAnimate" value="Animate">
105 <br>
106 <img id="logo" src="TestImage.gif" alt="Test Image">
107 </body>
108 </html>
```

How it Works?

1. jQuery provides built-in functions to create special visual effects such as show/hide, fade-in/fade-out and slide-in/slide-out. You can also create your own custom animation.
2. [TODO]

3.5 Debugging jQuery (Absolutely Important!!!)

I cannot stress more that having a proper debugging tool (and mind set) is indispensable in software development!!!

JavaScript/jQuery are interpreted instead of compiled. In other words, there is no compiler to check your syntax errors! Furthermore, during runtime, syntax errors are not reported. The error script simply returns false and stops working with no clue at all! You need a debugger to catch syntax

error for interpretive language. Catching logical errors without a debugger is even more challenging!

The popular client-side HTML/CSS/JavaScript debuggers are:

1. Chrome browser with Developer Tools.
2. Firefox with Firebug or Web Developer Tools.

On most browsers, you can press F12 to activate the Developer Tools. We call them F12 Developer Tool!

I strongly suggest that you trace through the jQuery statements in the above examples, by selecting the "script" panel. You often need to refresh (F5 or Ctrl-F5 to clear the cache) the page to get the correct script. Set breakpoint on the appropriate jQuery statements (take note that in Firebug, you can only set breakpoint on statements with green numbering). "Step Over (F10)" the statement, and watch the variables in the "Watch" panel. Under "Watch" panel, you can "add a new watch expression" to evaluate a jQuery or JavaScript expression, e.g., a jQuery selector.

To check the event handlers bound to an element, select the element (click on the "Select" icon and point at the element), then select the "Events" panel.

To debug the Ajax, watch the network traffic under the "Net" panel. You can select "xhr" for Ajax-specific network traffic.

Spent time to play around with the debugging tools. The more time you spend here, the less you will spend on staring at the screen wondering why it does not work and asking silly questions on problems caused by syntax errors such as a missing quote!

console.log()

Use `console.log(...)` to write message or object to the console for debugging. DON'T use `alert()` (which is annoying) or `document.write()` (which messes up your web page).

4. jQuery Basics

jQuery is an *extension* to JavaScript. In other words, it is JavaScript and follows the JavaScript syntax. Make sure you understand the JavaScript syntaxes and types, in particular, *functions* and *objects*. jQuery is quite easy to understand if you are proficient in JavaScript. You just have to trace through some jQuery operations using Firebug (or Web Developer Tools).

The jQuery API is available @ http://api.jquery.com/.

4.1 $(document).ready(*handler*)

Reference: The `.ready()` API.

In jQuery, we typically place our operations in *`handler`* under `$(document).ready(` *`handler`* `)`, which fires once the DOM tree is constructed, but before external resources (such as images) are loaded (equivalent to placing the jQuery scripts just before body closing tag `</body>`). This is more efficient than the JavaScript's `onload` handler (via `window.onload =` *`handler`* which fires after the document is completed downloaded). Furthermore, you can use multiple `.ready()`'s to register multiple handlers, which will be run in the order in which they were registered. JavaScript's `window.onload =` *`handler`* can be used only once.

The `.ready( handler )` takes an argument *`handler`*, which is most often an anonymous function; or a pre-defined function. The *`handler`* function has no argument. For example,

```
// 1. On "ready", callback an anonymous function (with no argument). Most
commonly-used
$(document).ready( function() {
   console.log('ready');
});

// 2. On "ready", callback a pre-defined function (with no argument)
$(document).ready(foo);
function foo() {   // function can be defined later
   console.log("foo");
}

// 3. Use a function variable - variable must be defined before used.
var bar = function() {   // function variable
   console.log("bar");
}
$(document).ready(bar);  // pass a function variable

// 4. This won't work!
//    .ready() takes a function object, not a statement!
// $(document).ready(alert("This won't work!"));
```

Shorthand $(*handler*)

The `$(document).ready( handler )` is so commonly-used, that there is a shorthand `$( handler )`.

4.2 DOM (Document Object Model)

[TODO]

4.3 jQuery Selector Function $() (or jQuery())

The real power of jQuery comes from its "Query" selector, used to search and retrieve matching DOM element(s). The jQuery selector function is denoted simply as `$()`, which usually takes a *selector* as the argument, and return a jQuery object. jQuery supports almost all the CSS 1 to CSS 3 selectors.

Take note that `$()` is an alias of `jQuery()` function. Recall that JavaScript's identifier must start with an alphabet, `'_'` or `'$'`. Hence, `$` could be used as an valid function variable name (which is one of the shortest possible name not beginning with a letter).

The `$()` returns a jQuery `object`, which is a wrapper over the selected DOM elements, plus more properties (e.g., `.length`) and methods (e.g., `.append()`, `.addClass()`, `.html()`).

`$()` could return one element (e.g., ID-Selector), more than one elements (e.g., Class-Selector and Tag-Selector); or zero elements (no matches found). For example,

```
1  <!DOCTYPE html>
2  <!-- JQSelectorTest.html -->
3  <html lang="en">
4  <head>
5  <meta charset="utf-8">
6  <title>Testing jQuery Selectors</title>
7  <script src="js/jquery-1.11.2.min.js"></script>
```

```
8  <script>
9  $( function() {   // Shorthand for .ready()
10    console.log($('#header'));  // ID-Selector
11    console.log($('p'));        // Tag-Selector
12    console.log($('.non-existence'));  // Class-Selector
13 });
14 </script>
15 </head>
16
17 <body>
18 <div id="header">
19 <p>Paragraph 1</p>
20 <p>Paragraph 2</p>
21 <p>Paragraph 3</p>
22 </div>
23 </body>
24 </html>
```

Study the Console Log:

1. The `$('#header')` ID-Selector selected one DOM element, wrap in a jQuery `object`, identified as `Object[div#header]`. The `.length` property is 1; and the DOM element is wrap under index 0 (i.e., `[0]`).
2. The `$('p')` Tag-Selector selected 3 DOM elements, wrap in a jQuery `object`, identified as `Object[p, p, p]`. The `.length` property is 3; and the DOM elements are wrap under `[0]`, `[1]` and `[2]`.
3. The `$('.non-existence')` Class-Selector selected zero elements, wrap in a jQuery `object`, identified as `Object[]`. The `.length` property is 0.
4. You can use the `.length` property of the resultant jQuery `object` to check the number of DOM elements matched.

Browse through the `object` returned by each of the above selectors.

4.4 Types of jQuery Selectors

General CSS Selectors

```
// T: Tag
// C: Class
// I: id
// A: Attribute name
// V: Attribute Value

*              // All elements
T              // All elements with tag-name T
T1,T2          // All elements with tag-names T1 or T2
T1 T2          // T2 nested under T1
T1>T2          // T2 is an immediate child of T1
T1+T2          // T2 immediately preceded by T1 (siblings)
.C             // Class of C
#I             // ID of I (id is unique)
T#I            // Tag-name T with ID of I
T#I.C          // Tag-name T with ID of I and class of C
T[A]           // Tag T with attribute A
T[A1][A2]      // Tag T with attribute A1 and A2
T[A="V"]       // Tag T with attribute A having value of V
T[A*="V"]      // Tag T with attribute A having value containing V
T[A^="V"]      // Tag T with attribute A having value starting with V
               // ^ denotes starting position as in regex
```

```
T[A$="V"]      // Tag T with attribute A having value ending with V
               // $ denotes ending position as in regex
T[A!="V"]      // Tag T with attribute A not having value of V
               //  or Tag T without attribute A
```

Special Positional Filter Selectors

jQuery adds many custom selectors on top of CSS selectors (marked with CSS). These selectors begins with a colon `':'`, similar to CSS's pseudo-class selector. Take note jQuery is zero-based (count from zero); while CSS is one-based (count from 1).

```
:first         // First matching element, e.g., p:first
:last          // Last matching element, e.g., li:last
:eq(n)         // nth matching element, e.g., a:eq(3), n begins from 0
:gt(n)         // greater than nth matching element, n begins from 0
:lt(n)         // less than nth matching element, n begins from 0
:even          // even matching element, e.g., tr:even for even rows, the top
row is row 0
:odd           // old matching element
:first-child     // (CSS) First child
:last-child      // (CSS) Last child
:only-child      // (CSS) no siblings
:nth-child(n)    // (CSS) nth child, n begins from 1
:nth-child(odd)  // (CSS) nth child, count from 1
:nth-child(even) // (CSS) nth child, count from 1
```

Filter Selectors

```
:not(S)        // (CSS) Not matching selector S
:has(S)        // containing element in selector S
:contain(text) // containing text (case sensitive)
```

Form Filter Selectors

```
:input         // input, select, textarea, button
:checkbox      // input[type=checkbox]
:radio         // input[type=radio]
:password      // input[type=password]
:button        // input[type=submit], input[type=reset], input[type=button],
button
:submit        // input[type=submit], button[type=submit]
:reset         // input[type=reset], button[type=reset]
:file          // input[type=file]
:image         // input[type=image]

:checked       // (CSS) checkbox and radio button in checked state
:selected      // (CSS) Selected <option> in <select>
:disabled      // (CSS) disable form elements
:enabled       // (CSS) enable form elements

:hidden        // All hidden elements (having CSS properties visibility:hidden)
:visible       // All visible elements (having CSS properties
visibility:visible)
```

Some Examples

```
p               // all <p> elements
#msg            // an element with id of msg (id is unique)
                // ID selector begins with #
.green          // all elements with class of green
                // Class selector begins with dot
p,a             // All <p> and <a> elements
p a             // <a> nested under <p>
```

```
p > a           // <a> which is an immediate child of <p>
ul > li > a     // <a> which is an immediate child of <li>
                //   which is an immediate child of <li>
input[required]     // <input> tags having the attribute "required"
input[type="text"]  // having the attribute and value
a[href^="http://"]  // <a> with attribute "href" starting with "http://"
                    // ^ denotes starting with (as in regex)
a[href?=".js"]      // <a> with attribute "href" ending with ".js"
                    // $ denotes ending with (as in regex)
a[href*="jQuery"]   // <a> with attribute "href" containing "jQuery"
```

Rule of thumb for choosing selectors:

1. Use ID-Selectors first, which is the most efficient, e.g., `$('#header')`, `$('#content')`.
2. Use ID, as ancestor of descendant selectors, e.g., `$('#header a')`, `$('#loginForm :text[name="foo"]')`.
3. Use `.find()` to locate the descendants, which serves the same function as descendant selector but slightly more efficient, e.g., `$('#header').find('img')`. The `.find()` is also used if you have already computed the results of the ancestor.
4. [TODO] more

4.5 Traversing the DOM Tree

- .find(*selector*): Apply *`selector`* to descendants of the matched elements.
- .filter(*selector*), .filter(*function*): Filter descendants based on the selector or applying the function.
- .parent():
- .closest(): nearest ancestor that matched.
- .siblings(): all siblings
- .next(): next sibling
- .nextAll(): all next siblings
- .previous(): previous sibling
- .previousAll(): all previous sibling
- .children():

4.6 Iterating Through All the Selected Elements

A jQuery selector may select zero or more DOM elements. The selected elements are wrapped inside an object, as `[0]`, `[1]`, ... etc. The `.length` property contains the number of elements selected. See the earlier EXAMPLE.

You can iterate through each of these selected elements via:

Implicit Iteration

For example, `$('p').append(...)` applies the `append()` function for each of the selected elements, in an implicit loop.

Explicit Iteration via .each(*function*) and $(this)

Suppose that you want to apply a series of operations to each of the selected elements, you could use `.each( function )` to iterate through all the selected elements. `.each()` takes a

function as its argument, which can be either an anonymous function, a pre-defined function, or a function variable. Within the function, you can use `$(this)` to refer to the element under operation.

For Example,

```
$( function() {  // shorthand for .ready()
   $('p').each( function() {  // Iterate through all selected elements
      console.log($(this));   // $(this) selects the element under operation
      $(this).append('<<< ');
      $(this).prepend(' >>>');
   });
});
```

The `.each()`'s argument *`function`* can take an optional argument, *`index`*, which counts the elements under operation, starts from 0. For example,

```
$('p').each( function(index) {
   ......
});
```

Besides `$(this)`, which refers to the jQuery object under operation, you can also use `this`, which refers to the DOM element under operation. That is,

```
$(this)[0] === this
```

You can apply jQuery methods (such as `.append()`, `.html()`) to `$(this)`, but not `this`. On the other hand, you can apply DOM operation to `this`, e.g., `this.id.substring(0,5)` (first five characters of the `id` attribute of the DOM element under operation).

Function Chaining

You can use *chaining* to apply a series of operation to each of the selected elements, as most of the jQuery functions return the jQuery object. For example,

```
$('p').append('before ').prepend(' after');
```

Function chaining is used extensively in jQuery, e.g., in a Ajax call.

Assign the Results of Selectors to a Variable

You can assign the result of selector (which is a set of elements) to a variable, and use the variable to perform subsequent operations. To signal the variable holds a selector result, it is named with a `'$'` prefix. For example,

```
var $para = $('p');                 // Select a set of elements
$para.prepend('Hi, ').append('!');  // Operate on EACH of the selected elements
```

4.7 Manipulating DOM Elements, HTML Attributes, CSS Properties

Recall that a jQuery selector function `$()` (or `jQuery()`) selects a set of DOM elements for manipulation. We shall use the term `innerHTML` (a term used in JavaScript) to refer to the contents excluding the opening and closing tags.

Manipulating Contents of the Selected Elements

Function	Description	Example

`.html()`	Get the innerHTML, including nested elements.
`.html(`*`newContent`*`)`	Replace the innerHTML with the *`newContent`*, which may include nested element.
`.text()`	Get the combined text, including nested elements, but excluding the tags.
`.text(`*`newText`*`)`	Replace the content with the *`newText`*, which cannot include tags.
`.append(`*`value`*`)`	Append the *`value`* to the end of innerHTML. *`value`* may include nested element.
`.prepend(`*`value`*`)`	
`.before(`*`value`*`)`	Add the *`value`* before the opening tag of this element. *`value`* is likely an HTML element
`.after(`*`value`*`)`	
`.remove()`	Remove the selected element
`.empty()`	Remove the innerHTML.
`.replaceWith(`*`value`*`)`	Replace this element with the *`value`*.
`.wrap(`*`tag`*`)`	Wrap the selected elements with the given *`tag`*.
`.warpInner(`*`tag`*`)`	Wrap the innerHTML with the given *`tag`*.
`.unwrap()`	Unwrap the element, by removing the outermost tag.

Reversing the Source and Target

In the above table, we use `$(selector)` to select a set of DOM elements, and apply the desired operations. For example,

1. `$(`*`selector`*`).before(`*`HtmlElement`*`)`: insert before and outside the selected elements.
2. `$(`*`selector`*`).after(`*`HtmlElement`*`)`: insert after and outside the selected elements.
3. `$(`*`selector`*`).append(`*`HtmlElement`*`)`: insert before and inside the selected elements.
4. `$(`*`selector`*`).perpend(`*`HtmlElement`*`)`: insert after and inside the selected elements.

The selector function `$()` is overloaded, such that `$(HtmlElement)` creates the given HTML element. We can then place the created element on the DOM tree via the following functions, which reverse the source and target:

1. `$(`*`HtmlElement`*`).insertBefore(`*`selector`*`)`: insert before and outside the selected elements.
2. `$(`*`HtmlElement`*`).insertAfter(`*`selector`*`)`: insert after and outside the selected elements.
3. `$(`*`HtmlElement`*`).appendTo(`*`selector`*`)`: insert after but inside the selected elements.
4. `$(`*`HtmlElement`*`).prependTo(`*`selector`*`)`: insert before but inside the selected elements.

Manipulating CSS Properties for Styling

A CSS style contains a list of property `name:value` pairs, (e.g., `color:red`; `background-color:black`). We can achieve various presentation effects by manipulating CSS properties attached to a DOM element. In addition, the `class` attribute is used extensively for applying CSS styles.

We can:

1. *directly* add/remove CSS properties attached to selected DOM elements, via `.css()`; or
2. *indirectly* add/remove values from the `class` attribute, via `.addClass()` or `.removeClass()`.

Function	Description	Example
`.addClass(`*`value`*`)`	Add *`value`* (*classname*) to the HTML `class` attribute.	
`.removeClass(`*`value`*`)`	Remove *`value`* (*classname*) from the HTML `class` attribute.	
`.toggleClass(`*`value`*`)`	Add if *`value`* does not exist; otherwise, remove.	
`.css(`*`property`*`)`	Get the value of one CSS property.	
`.css([`*`property1, property2`*`])`	Get the value of the multiple CSS properties. Return an associative array of *`property`*`:`*`value`*.	
`.css(`*`property, value`*`)`	Set the value of one CSS property.	
`.css({`*`property`*`1:`*`value`*`1, `*`property`*`2:`*`value`*`2, ...})`	Set multiple CSS properties, by passing an associative array of *`property`*`:`*`value`* pairs.	`$('p').css({'color':'red', 'background-color':'blue'})`

Manipulating HTML Attributes

An HTML element may contain attributes (e.g., `id`, `class` and many others). We can add/remove attribute via `.attr()`, `.removeAttr()`.

Beside attributes, a DOM object has properties such as `defaultChecked`, `defaultSelected`, `selectedIndex`, `tagName`, `nodeName`, `nodeType` and `ownerDocument`. These properties should be manipulated via `.prop()` and `.removeProp()`. The `.prop()` should be used for boolean properties of input elements such as `enabled/disabled`, `checked` and `selected`.

Function	Description	Example
`.attr(`*`attribute`*`)`	Get the value of HTML attribute.	
`.attr(`*`attribute, value`*`)`	Set the *`value`* of the HTML attribue.	
`.attr({`*`attribute1`*`:`*`value1, attribute2`*`:`*`value2, ...`*`})`	Set the value of multiple HTML attribute by passing an associative array of *`attribute`*`:`*`value`* pairs.	

`.removeAttr(`*`attritbue`*`)`	Remove the HTML attribute.
`.prop(`*`property`*`)`	
`.prop(`*`property`*`, `*`value`*`)`	
`.prop({`*`property1`*`:`*`value1`*`, `*`property2`*`:`*`value2`*`, ...})`	
`.removeProp()`	

4.8 Handling Events

JavaScript vs. jQuery Event Handling

In JavaScript, can we bind the event handler in two ways:

1. Via the attribute `onxxx`, for example,

   ```
   <body onload="init()" >
   <body onload="init1();init2()" >
   <form onsubmit="validateForm(this)" >
   ```

   You can place a series of JavaScript statements (separated by semi-colon).
   This is not recommended nowadays, as it mix the behavior programming code and the contents.
2. Via the JavaScript, e.g.,

   ```
   window.onload = init;   // Named function, no parentheses
   function init() { ..... }

   window.onload = function() { ..... };  // anonymous function

   document.document.getElementById('#theForm').onsubmit = validateForm;
   function validateForm() { ..... }
   ```

jQuery is much simpler, and always separated from the HTML codes. Anonymous functions (instead of named functions) are used extensively. E.g.,

```
// Bind handler to event
$('#theForm').on('submit', function() { .... });  // Anonymous function
$('#theForm').submit(function() { .... });        // Shorthand, same as above

$('#theForm').on('submit', validateForm);    // Named function, no parentheses
function validateForm() { ..... };

$('#btn').on('click', function() { .... });
$('#btn').click(function() { .... });  // same as above
```

jQuery Events

The jQuery events are:

mouse	`click`, `dblClick`	Single mouse-click or double mouse-click
	`mousedown`, `mouseup`	Pressing/Releasing the mouse button
	`mouseover`, `mouseout`	Mouse pointer move into or out of the element.
	`mousemove`	Mouse pointer moves inside the element. It report the (x, y) of the mouse pointer.
`document`	`ready`	DOM tree constructed. The external resources (such as images)

		may not be loaded.
	`load`	Fully loaded the web page including all its referenced resources (such as images).
	`unload`	Click a link to another page; close the window/tab. Run before to do housekeeping.
`window`	`resize`	Resize the browser's window
	`scroll`	Scroll the web page
	`submit`	Submitting form (`<input type="submit">`)
`form`	`reset`	Reset form (`<input type="reset">`)
	`change`	Change selection (checkbox, radio, drop-down menu)
	`focus`, `blur`	The input element gain/loss the field focus.
	`keypress`	Typing a key
`key`	`keyup`, `keydown`	Press/Release a key. To get the key character, use: `var key = String.fromCharCode(event.which);`

Most of the jQuery codes is dealing with programming event handlers for selected elements. The steps are:

1. Select the *source* elements, e.g., `$(':submit')` for the submit button.
2. Identify the event, e.g., `click`.
3. Write the event handler using a built-in or pre-defined function (e.g., `$(':submit').click(alert(...))`; or an anonymous function to program a series of operations, e.g., `$(':submit').click(function() {...})`.

The .hover() function

The `mouseover` and `mouseout` events are commonly used together. Instead of writing two handlers, jQuery provides a combined `.hover(mouseoverFunction, mouseoutFunction)`, which takes two function arguments: `mouseover` handler and `mouseout` handler. For example,

```
$('#menu').hover(
   function() {   // mouseover
      $('#submenu').show();
   },
   function() {   // mouseout
      $('#submenu').hide();
   }
);
```

The event Object

The event handling functions accept an optional argument of an `event` object. The `event` object captures information about the event, e.g., mouse (x, y), the key pressed, etc. For example,

```
$('#box').click(function(evt) {
   var xPos = evt.pageX;
   var yPos = evt.pageY;
   alert('X:' + xPos + ' Y:' + yPos);
});
```

event's Properties

The `event` object has these properties:

`target`	The source element that triggered the event.
`pageX, pageY`	`(x, y)` of the browser's window.
`screenX, screenY`	`(x, y)` of the monitor.
`which`	The numeric code of the key typed. Use `String.fromCharCode(evt.which)` to get the character typed.
`shiftkey`	true if shift key is pressed.
`data`	data pass over from `.on(`*`events, selector, data, handler`*`)`.

Preventing Default Action (or Normal Behavior)

As an example, the default action of click a link is to bring you to a new page. You could prevent the default action by either:

1. Invoke `event.preventDefault()` function in the event handler.
2. Return false from the event handler.

Event Propagation

In jQuery, an event (e.g., `mouseover`) is sent to the most specific element, and then *bubbles up* through its ancestors. You can use `event.target` to find the element first receiving the event.

To stop event propagation, use `event.stopPropagation()`.

To prevent the default action of an event (e.g., following the hyperlink in `<a>`; submitting a form when clicking submit button), use `event.preventDefault()`.

Binding Event Handler .on(events[, selector] [, data], handler)

The `.on()` binds a handler to one or more events.

- `events`: one or more events, separated by space.
- `handler`: event handling function in the form of `Function(Event eventObject)`.
- `selector`: a select to filter the descendants of the selected elements that trigger the event. Use for event delegation for dynamically generated descendant elements.
- `data`: data to be passed to `event.data`, where `event` is the optional argument of the handler function.

For example,

```
[TODO]
```

Delegating Event via .on()

Suppose that you have a `<table>` with `<tr>` dynamically generated, you cannot attach handler directly to `<tr>`'s, as they have yet to be created. Instead, you can attach the event handler to `<table>` and then delegate it to `<tr>`, via `.on()` optional argument `selector`. If `selector` is present and not `null`, it will be used to filter the descendants of the element operated. Instead of triggering on the element operated, it will trigger on the matched descendants.

For example,

[TODO]

Unbinding Event Handlers .off()

Counterpart of `.on()`, used to remove an event handler.

You can use `.off(event)` to remove the handler for that *`event`*. You can use no-arg `.off()` to remove all event handlers for the selected elements.

Shorthand Binding .click(), .submit(), ...

These are shorthand for `.on(event, handler)`, for the respective events.

Triggering an Event .trigger(*event*)

Simulating triggering of an event.

5. Animation and Special Effects

jQuery Built-in Functions for Special Visual Effects

	Function	Description	Example
Hide/ Show	`.hide()`	Hide the selected elements.	
	`.show()`	Show the selected elements.	
	`.toggle()`	If the element is visible, hide it; otherwise, show it.	
Fade	`.fadeIn()`	Show a hidden element by fading into view.	
	`.fadeOut()`	Hide a visible element by fading out of view.	
	`.fadeToggle()`	If the element is visible, fade-out; Otherwise, fade-in.	
	`.fadeTo()`	Fade an image to a specific opacity - between 0 (totally transparent) and 1 (totally opaque).	
Slide	`.slideDown()`	Show a hidden element by sliding down into view.	
	`.slideUp()`	Hide a visible element by sliding up out of view.	
	`.slideToggle()`	If the element is hidden, slide-down into view; otherwise, slide-up out of view.	

Most of the special effect functions take two optional arguments:

1. The first argument specifies the speed, in special literals of `'fast'` (200 msec), `'normal'` (400 msec), or `'slow'` (600 msec); or a time value in milliseconds.
2. The second argument is a *callback function* to be executed after the effect is completed.

Custom Animations via .animate()

The `.animate()` allows you to animate any CSS property that accepts numeric values such as `px`, `em`, or `percent`, e.g., font size, position, margin, padding, border, and opacity.

The `.animate()` takes up to 4 arguments:

1. an associative array of CSS properties.
2. optional speed in milliseconds.
3. optional easing type.
4. optional callback function, after the effect completed.

For example, suppose `#logo` selects an image, the image will move to the new margins, with the new opacity.

```
$(#logo).animate(
   {  'margin-left': '30px',   // CSS properties
      'margin-top': '20px',
      'opacity': 0.5
   },
   1000   // Speed in milliseconds
);
```

6. Handling HTML (Web) Form

Form-related Filter Selectors

jQuery Filter Selector	Same as Selector	HTML Element
:input	input, select, textarea, button	<input> <select>..</select> <textarea>..</textarea> <button>..</button>
:submit	input[type=submit], button[type=submit]	<input type="submit"> <button type="submit">..</button>
:text	input[type=text]	<input type="text">
:password	input[type=password]	<input type="password">
:checkbox	input[type=checkbox]	<input type="checkbox">
:radio	input[type=radio]	<input type="radio">
:file	input[type=file]	<input type="file">
:image	input[type=image]	<input type="image">
:button	input[type=submit], input[type=reset], input[type=button], button	<input type="submit"> <input type="reset"> <input type="button"> <button>..</button>
:reset	input[type=reset], button[type=reset]	<input type="reset"> <button type="reset">..</button>
:checked		<input type="checkbox|radio" checked>
:selected		<select><option selected>..</option>...</select>
:enabled		
:disabled		
:hidden		CSS property visibility:hidden

`:visible`	CSS property `visibility:visiable`

Form-Related Event Handler

`.submit()`	Submitting form by clicking the `<input type="submit">`, `<input type="image">`, or `<button type="submit">..</button>`
`.reset()`	Reset form (`<input type="reset">`, or `<button type="reset">..</button>`)
`.change()`	Value changed. For <select>, <input type="checkbox"> and <input type="radio">, the event is fired immediately when a selection is made. For `<input type="text">`, `<textarea>`, the event is fired after the input element lose focus.
`.click()`	In web form, used for submit/reset buttons `<input type="submit">`, `<input type="reset">`, `<button type="submit">`, `<button type="reset">`.
`.focus()`	The input element gains the field focus. Eg. "tab" into the input element.
`.blur()`	The input element loses the focus.

Get input value: .val()

- .val(): Getter current value of the first element in the set of matched elements. Used of getting the value of `<input>`'s types of `text`, `checkbox`, `radio`, `<select>` and `<textarea>`. For example,

```
// Get the value of text
$('#formID :text[name="foo"]').val();  // <form id="formID"><input
type="text" name="foo">

// Get the value from <select>
$('#selectID:selected').val();        // <select id="selecctID">'s
selected value
$('select[name="foo"]').val();        // <select name="foo">'s selected
value
$('#formID :selected').val()          // <form id="formID"><select ...>

// Get the value of checked checkbox
$('#formID input:checkbox[name="foo"]:checked').val()   // <form
id="formID"><input type="checkbox" name="foo">
$('#formID input[type="checkbox"][name="foo"]').val()

// Get the value of the checked radio
$('#formID input:radio[name="foo"]:checked').val()      // <form
id="formID"><input type="radio" name="foo">
```

- .val(*value*): Set the value of each of the selected elements.

The .submit() Event Handler

The `submit` event can be triggered by clicking the `<input type="submit">`, `<input type="image">`, or `<button type="submit">...</button>`. For example,

```
$('#formID').submit( function(evt) {  // submit event handler
   ......
   ......
   // return false to prevent form submission, or evt.preventDefault()
});
```

You can also use the generic `.on('submit', handler)` for `.submit(handler)`.

7. Ajax

7.1 What is Ajax?

The term "Ajax" was first mentioned in 2005, originally stands for "*Asynchronous JavaScript and XML*". It has since gone beyond XML, and currently serves as a client-side technology for JavaScript to transfer data between browser and server in the *background asynchronously*, so as to provide better responsiveness, without locking down the browser. The classical example of Ajax is google map, which allows you to carry out tasks while loading the maps.

Ajax involves:

1. JavaScript as the engine.
2. a JavaScript object called `XMLHttpRequest`, which does all the works. It sends request to the web server, waits for the response, processes the response, and updates some parts of the page by manipulating the DOM, based on the response received.
3. text data in the form of plain text, HTML, XML or JSON (JavaScript Object Notation).

Why Ajax (Instead of an Ordinary HTTP Request)?

1. Better responsiveness: Without asynchronous Ajax request, the browser will freeze (and hang) while processing the request. On the other hand, with the asynchronous Ajax request in the background, the browser (and JavaScript engine) do not have to wait for the response and can process other tasks.
2. No reloading of the entire page: You can update a portion of the page, instead of refreshing the entire page.

Debugging Ajax with Firebug (or Web Developer Tools) under FireFox

Under Firefox/Firebug, you could view all the network traffic under the "Net" tab, including the request/response messages. To view only Ajax messages, select "XHR" (XMLHttpRequest).

7.2 Ajax with jQuery

You can write Ajax using raw JavaScript. However, jQuery makes it much simpler.

The $.ajax(*settings*) or $.ajax(*url*, *settings*)

Used for sending an Ajax request. The *settings* is an object of key-value pairs. The frequently-used keys are:

- *`url`*: The request URL, which can be placed outside the *`settings`* in the latter form.
- *`type`*: GET or POST.
- *`data`*: Request parameters (name=value pairs). Can be expressed as an object (e.g., `{name:"peter", msg:"hello"}`), or query string (e.g., `"name=peter&msg=hello"`).
- *`dataType`*: Expected response data type, such as `text`, `xml`, `json`, `script` or `html`.
- *`headers`*: an object for request header key-value pairs. The header `X-Requested-With:XMLHttpRequest` is always added.

Ajax request, by default, is asynchronous. In other words, once the `.ajax()` is issued, the script will not wait for the response, but continue into the next statement, so as not to lock up and freeze the screen.

NOTE: `$` is a shorthand (alias) for the `jQuery` object. `$()` is an alias for `jQuery()` function for Selector. `$.ajax()` is a global function (similar to class method in an OO language).

.done(), fail(), .always()

You can chain `.done()`, `.fail()`, `.always()` after `$.ajax()` or all the convenience functions (to be discussed later), other than `.load()`.

- `.done( function(`*`responseText`*`) )`: run if the Ajax request succeeds (i.e., response code 200).
- `.fail( function(`*`jqXHR, textStatus, error`*`) )`: run if the Ajax request fails (i.e., response code is NOT 200).
  - *`jqXHR`* is the jQuery wrapper of XMLHttpRequest object. You can get the response text via *`jqXHR`*`.responseText`; the response status error code (e.g., 404, 500) via *`jqXHR`*`.status`; and the response status text (e.g., "Not Found", "Internal Server Error") via *`jqXHR`*`.statusText`.
  - *`textStatus`* is simply "error" for `.fail()`.
  - *`error`* is the HTTP response status text (e.g., "Not Found", "Internal Server Error").
- `.always( function() )`: always run after the `.done()` or `.fail()`.

Note: the `.done()`, `.fail()`, `.always()` deprecate the `success`, `error` and `complete` keys in `.ajax()`'s *`settings`*.

Examples

[TODO]

Serialize Form data .serialize()

You can use `$('#form').serialize()` to extract all the input request parameters and URL-encode into a query string.

Server Checking for Ajax Request

An Ajax request has a request header "`X-Requested-With: XMLHttpRequest`". Your server-side program can check if the request is an Ajax via this header. For example, in PHP:

```
<?php
$ajax = isset($_SERVER['HTTP_X_REQUESTED_WITH'])
        && $_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest';
......
?>
```

Security Considerations

- To prevent XSS (Cross-Site Scripting) attack, the XMLHttpRequest object can only request data from the original server that serve the page.
- Be careful in downloading script and running the script!

Loading HTML into Element $(*selector*).load(*url*[, data][, complete])

Load data from the server and place the return HTML inside the matched element. The `.load()` is a shorthand method for `.ajax()`. It is the simplest Ajax method to load data from server.

- *`url`*: request URL.

- *data*: Request parameters (name=value pairs). Take an object (e.g., `{name:'peter', message:'hello'}`), or string (e.g., `"name=peter&message=hello"`). By default, `.load()` uses POST method for object; and GET method for string.
- *complete*: A function to be called back when the request completes, in the form of `Function(responseText, textStatus, jqXHR)`. You can get the response status code (e.g., 404, 500) vai `jqXHR.status`; and the status text (e.g., "Not Found", "Internal Server Error") via `jqXHR.statusText`.

The `.load()` allows you to load *portion* of a document by appending a selector after the *url*, seperating by space, e.g.,

```
$('#divResult').load('ajaxTest.html #contentDiv > p');
```

The jQuery parses the returned HTML document to locate the selector and discards the rest of the document.

To check for error:

```
$( "#success" ).load( "test.php", function( response, textStatus, jqXHR ) {
   if ( textStatus === "error" ) {
      $( "#error" ).html( "Error: " + jqXHR.status + " " + jqXHR.statusText );
            // jqXHR.status gives the status code, e.g., 404, 500
            // jqXHR.statusText gives the text, e.g., "Not Found", "Internal
Server Error".
   }
});
```

Loading JSON Data and $.getJSON()

JSON (JavaScript Object Notation) is a lightweight textual data-interchange format. Recall that a JavaScript object is a set of key-value pair, written as `{ 'key1':'value1', 'key2':'value2', ...}`. On the other hand, a JavaScript array has an implicit numeric key, written as `['item1', 123, 'item3', ...]`. JSON uses these two formats to provide a concise notation (instead of the bulky XML). For example,

```
{
   "key1" : "value1",
   "key2" : ["stringItem1", 1234, "stringItem3"],
   "key3" : "value3"
}
```

In JSON, all object keys and string literals, must be enclosed with *double quote*.

The `$.getJSON(`*url*`[, data] [, `*success*`])` global function fetches JSON data and converts it to an JavaScript object. It is a shorthand method for `.ajax()` where `dataType:json`.

JavaScript's JSON.parse() and jQuery's $.parseJSON()

You can use JavaScript's function `JSON.parse(`*text*`)` to parse a JSON-formatted text into an object. For example,

```
var jsonText = '{'
             + '"key1" : "value1",'
             + '"key2" : ["stringItem1", 1234, "stringItem3"],'
             + '"key3" : "value3"'
```

```
            + '}';
console.log(JSON.parse(jsonText));
      // Object { key1="value1",  key2=[3],  key3="value3"}
```

jQuery's `$.parseJSON()` is a wrapper for JavaScript's `JSON.parse()`.

JavaScript's JSON.stringify()

The counterpart of `JSON.parse()`, which converts a JavaScript value or object to a JSON string. There is no equivalence in jQuery.

Server Returning JSON Data

The server should set a response header "`Content-Type: application/json`". For example, in PHP:

```
<?php
header('Content-Type: application/json');
echo json_encode($jsonText);
?>
```

Loading Script and $.getScript(*url*[, *success*])

We can use global function `$.getScript(`*`url`*`)` to load and execute a JavaScript. It is a shorthand method for `.ajax()` where `dataType:script`.

$.get(url[, data][, success][, dataType]) and $.post(url[, data][, success][, dataType])

Send an HTTP GET or POST request. They are shorthand methods for `.ajax()`.

- *`dataType`*: expected data type from the server, e.g., `xml`, `json`, `script`, or `html`.

Providing Feedbacks for Ajax Requests: .ajaxStart(), .ajaxStop(), .ajaxComplete()

- `$(document).ajaxStart( `*`function`*` )`: run when the first Ajax request begins.
- `$(document).ajaxComplete( `*`function`*` )`: runs whenever an Ajax request completes.
- `$(document).ajaxStop( `*`function`*` )`: runs after ALL Ajax requests completes.

7.3 Ajax using Raw JavaScript (Not Recommended)

I strongly suggest that you use jQuery for Ajax, which is much simpler than the raw JavaScript. However, I keep this old section here for completeness and explanation of the underlying raw JavaScript object.

All Ajax operations are carried out through a JavaScript object called `XMLHttpRequest`.

Constructing an XMLHttpRequest Object

To create a `XMLHttpRequest` object, invoke the constructor:

```
var xhr = new XMLHttpRequest();
```

XMLHttpRequest Properties and Methods

The `XMLHttpRequest` contains a set of properties (or variables) and methods.

The properties are:

- readyState: 0: uninitialized; 1: loading; 2: Loaded; 3: Interactive; and 4: completed.
- onreadystatechange: for specifying an event handler for the change in `readyState`.
- responseText and responseXML: The response returned by the server in text and XML format.
- status and statusText: The status code (e.g., 200, 404) and status message (e.g., OK, Page not found).

The methods are:

- open(*method*, *url*, *isAsyn*): Open an HTTP connection with the specified *`method`* (GET or POST), *`url`*, and whether the request should be handled asynchronously (`true` or `false`). Asynchronous request run in the background and does not freeze the browser. Using async request is always recommended, except short and small request.
- setRequestHeader(*param*, *value*): Set the request header for the given param=value pair.
- send(*param*s): send the GET/POST key=value parameters to the server.
- abort(): abort the current request.
- getAllResponseHeader(): returns all headers as a string.
- getResponseHeader(*param*): returns the value of given *`param`* as a string.

EXAMPLE 1: Ajax Request to HTTP Server for Plain Text

In this example, we use Ajax to send a POST request asynchronously to request for a text file.

"**JSAjaxEx1.html**"

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Hello, Ajax</title>
</head>

<body>
<h2>Hello, Ajax</h2>
<!-- Placeholder for Ajax response text -->
<div id="ajaxResponse"></div>
<a href="test" onclick="loadAjaxText(); return false">SEND</a>

<script>
// Load Ajax responseText into element with id="ajaxResponse"
function loadAjaxText() {
   // Construct a XMLHttpRequest object
   var xhr = new XMLHttpRequest();
   // Set up the readyState change event handler
   xhr.onreadystatechange = function() {
      if ((this.readyState === 4) && (this.status === 200)) {
         if (this.responseText !== null) {
            // Load the responseText into element with id="ajaxResponse"
            var outElm = document.getElementById(ajaxResponse);
            outElm.innerHTML = this.responseText;
         }
      }
   }
   // Open an asynchronous POST connection and send request
   xhr.open("POST", "HelloAjax.txt", true);
   xhr.send();
```

```
32 }
33 </script>
34 </body>
35 </html>
```

"HelloAjax.txt"

```
This is the Response Text of Ajax Request!!!
```

You need to run this example under an HTTP server (e.g., Apache) as it sends an HTTP request.

How it Works?

The client-side HTML contains a `<div>`, to be used as placeholder for the response text. It also contains a hyperlink to trigger the Ajax request (via the function `loadAjaxText()`).

The steps in using Ajax, in function `loadAjaxText()` are:

1. Allocate an `XMLHttpRequest` object, via `new XMLHttpRequest()`.
2. Open a connection to the server, via `open("GET|POST", url, isAync)` method. In this case, we request for a text file called "`HelloAjax.txt`".
3. Invoke `send()` to send the request to the server.
4. Provide the event handler `onreadystatechange` to handle the `readyState` change event. The `readyState` of 4 indicates Ajax transfer completed; `status` of 200 (`statusCode` of OK) indicates successful HTTP request and the response message can be retrieved from the `xhr.responseText`. In this example, we load the `responseText` into the element with `id="ajaxResponse"`.

EXAMPLE 2: Ajax Request with POST Parameters to PHP HTTP Server

In this example, we trigger an Ajax POST request to a PHP page to obtain dynamic response.

"JSAjaxEx2.html"

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="utf-8">
5  <title>Hello, Ajax from PHP</title>
6  </head>
7
8  <body>
9  <h2>Hello, Ajax from PHP</h2>
10 <textarea id="inText" name="inText" cols="40" rows="5">Enter your text here...</textare
11 <textarea id="ajaxResponse" name="ajaxResponse" cols="40" rows="5"></textarea><br>
12 <a href="test" onclick="loadAjaxText(); return false">SEND</a>
13
14 <script>
15 // Append Ajax responseText into element with id="ajaxResponse"
16 function loadAjaxText() {
17    // Construct an XMLHttpRequest object
18    var xhr=new XMLHttpRequest();
19    // Set up the readyState change event handler
20    xhr.onreadystatechange = function() {
21       if ((this.readyState === 4) && (this.status === 200)) {
22          if (this.responseText !== null) {
23             var ajaxElm = document.getElementById('ajaxResponse');
```

```
24             ajaxElm.innerHTML = this.responseText + ajaxElm.innerHTML; // append in fro
25          }
26       }
27    }
28    // Open an asynchronous POST connection and send request
29    xhr.open("POST", "HelloAjax.php", true);
30    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
31    var inTextElm = document.getElementById('inText');
32    xhr.send("inText=" + inTextElm.value);
33 }
34 </script>
35 </body>
36 </html>
```

The client-side HTML contains two `<textarea>`s, for the input and output respectively. It also contains a hyperlink to trigger the Ajax request (via the JavaScript function `loadText()`).

The steps in using Ajax, in JavaScript function `loadText()` are:

1. Allocate an `XMLHttpRequest` object, via `new XMLHttpRequest()`.
2. Open a GET or POST connection to the server, via `open("GET|POST", `*`url`*`, `*`isAync`*`)` method.
3. Set the HTTP request headers, if needed. For example, you need to set the HTTP request header "`Content-type`" to "`application/x-www-form-urlencoded`" if you are sending POST data.
4. Invoke `send(`*`data`*`)` to send the POST request with query parameters to the server.
5. Provide the event handler `onreadystatechange()` for the `readyState` change event. The `readyState` of 4 indicates Ajax transfer completed; `status` of 200 (`statusCode` of OK) indicates successful request; the response message can be retrieved from the `xhr.responseText`.

Server-side Program: HelloAjax.php

```
1 <?php
2 // Echo back the POST parameter inText=xxx
3 if (isset($_POST['inText'])) {
4   echo ">> {$_POST['inText']}\n";
5 }
6 ?>
```

The server-side program `HelloAjax.php` handles a POST request with parameter `inText=`*`message`*, and echoes back the input message.

EXAMPLE 3: Ajax Request for XML Document

"JSAjaxEx3.html"

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="utf-8">
5  <title>Hello, Ajax for XML</title>
6  </head>
7
8  <body>
```

```
9  <h2>Hello, Ajax for XML</h2>
10 <!-- holder for Ajax response text -->
11 <div id="ajaxResponse"></div>
12 <a href="test" onclick="loadText(); return false">SEND</a>
13
14 <script>
15 function loadText() {
16    // Construct a XMLHttpRequest object
17    var xhr=new XMLHttpRequest();
18    xhr.onreadystatechange = function() {
19       if ((this.readyState === 4) && (this.status === 200)) {
20          if (this.responseXML !== null) {
21             var xmlDoc = this.responseXML;
22             var txt = "";
23             // Retrieve all the "title" elements from the XML document
24             var titleElms = xmlDoc.getElementsByTagName("title");
25             for (i = 0; i < titleElms.length; i++) {
26                txt += titleElms[i].childNodes[0].nodeValue + "<br>";
27             }
28             document.getElementById("ajaxResponse").innerHTML = txt;
29          }
30       }
31    }
32    xhr.open("POST", "HelloAjax.xml", true);
33    xhr.send();
34 }
35 </script>
36 </body>
37 </html>
```

The XML File: HelloAjax.xml

```
<bookstore>
  <book>
    <title>Java For Dummies</title>
    <author>Tan Ah Teck</author>
  </book>
  <book>
    <title>Java For More Dummies</title>
    <author>Tan Ah Teck</author>
  </book>
</bookstore>
```

8. Testing JavaScript/jQuery with QUnit

QUnit @ https://qunitjs.com/ is a unit testing framework for JavaScript. It is used in unit testing for jQuery, jQuery UI, jQuery Mobile, and others.

8.1 Setting Up

Down the latest `qunit-1.xx.x.js` and `qunit.1.xx.x.css` from https://qunitjs.com/ and place them in appropriate folders. Link the `js` and `css` in your document. Alternatively, you can link to one of the CDN (Content Distribution Network). QUnit does NOT require jQuery, i.e., there is no need to load the `jquery.js`, unless you are testing codes with jQuery.

"QUnitSetup.html"

```
1  <!DOCTYPE html>
2  <!-- QUnitSetup.html -->
3  <html lang="en">
```

```
4  <head>
5  <meta charset="utf-8">
6  <title>QUnit Setting Up</title>
7  <link rel="stylesheet" href="css/qunit-1.18.0.css">
8  <script src="js/qunit-1.18.0.js"></script>
9  <script src="test/QunitSetup.js"></script>
10 </head>
11 <body>
12   <div id="qunit"></div>
13   <div id="qunit-fixture"></div>
14 </body>
15 </html>
```

1. In Line 7, we link to the QUnit's CSS file.
2. In Line 8, we link to the QUnit's JavaScript file.
3. In Line 9, we link to our test script. It is recommended to place the test script in an external file, rather than embedded; and in a separate folder (say `test`).
4. The `<div>` in Line 12-13 are used by QUnit to display the test results.

"QUnitSetup.js"

Our first test script is as follows:

```
1  QUnit.test( "Hello test", function( assert ) {
2     assert.ok( 1 == "1", "Expecting true. This test shall PASS!" );
3     assert.ok( 1 === "1", "Expecting false. This test shall FAIL, for illustration!" );
4     var message = "hello";
5     assert.equal(message, "hello", "Expecting message to be hello");
6           // actual, expected, message. Use == for comparison
7     var number = "88";
8     assert.deepEqual(number, 88, "Expecting a number type of value 88");
9           // Use === for comparison, i.e., same type and value
10 });
```

Try running the tests by loading the HTML file.

We have a *test suite* called "Hello test", consisting of four *tests*:

1. Test 1 asserts the expression `1=="1"` to be `true`, which evaluates to `true`. This test passes.
2. Test 2 asserts the expression `1==="1"` to be `true`, but it evaluates to `false`. This test fails. Take note that this is not the correct way of writing test, I do it for illustrating failed test.
3. Test 3 asserts that the actual value in first argument is equal (using `==`) to the expected value in the second argument. This test passes.
4. In test 4, strictly equal (`===`), which compares both the type and value is used. This test fails.

Read:

1. QUnit's "Introduction to Unit Testing" @ http://qunitjs.com/intro/.
2. QUnit Cookbook @ http://qunitjs.com/cookbook/.

8.3 QUnit Basics

You need not place the test under JavaScript's `onload` or jQuery `ready()`, as by default, QUnit waits until the page is loaded before running its tests.

Assertions

The assertions available in QUnit are:

- assert.ok(*truth*[, *message*]): The test passes if *`truth`* evaluates to `true`. The optional string *`message`* is for display in the test results.
- assert.equal(*actual*, *expected*[,]): The test passes if *`actual`* is equal (==) to *`expected`*. Compare with `assert.ok()`, `assert.equal()` displays the actual and expected values if the test fails.
- assert.deepEqual(*actual*, *expected*[,]): Similar to `assert.equal()` but uses strictly equal (===), (i.e., same type and value) to compare the actual and expected values.
- assert.expect(numAsserts): to state the number of assertions in the test suite. [TODO] Example.
- assert.async(): for asynchronous operations such as user actions or Ajax calls, which returns a "done" function that should be called when the test operation has completed. [TODO] Example.

Module of Tests

QUnit provides two levels of structure to organize tests. A `QUnit.test()` test suite groups a set of tests; while `QUnit.module()` groups a set of `QUnit.test()`. For example,

```
QUint.module( "Module A" );
QUnit.test( "Module A Test 1", .... );
QUnit.test( "Module A Test 2", .... );
QUnit.test( "Module A Test 3", .... );
QUint.module( "Module B" );
QUnit.test( "Module B Test 1", .... );
QUnit.test( "Module B Test 2", .... );
```

You can use the module name to selectively run a module.