Client Side Libraries
Week 10 Lecture

Outline

- Intro to jQuery
  - Selectors
  - Event handler and DOM man
  - Ajax requests
- Integrate jQuery with Expressjs Application

Revisits Client Side Technologies

- Web client is not a pure passive receiver of data sent from the server
- Modern client has lots of interactive features to make it resemble desktop GUI
  - HTML5
  - CSS3
  - JavaScript
- Many client side JavaScript libraries
  - jQuery
  - Specialized libraries, e.g. D3.js, various google libraries
- Client side “scripting” becomes real application development with its own model, view and controller
  - AngularJS framework
  - Backbone MVC framework

jQuery

- jQuery is a lightweight JavaScript library
  - Provides methods to wrap common JavaScript tasks
    - Selecting elements
    - Register element’s event handler
    - Managing asynchronized request
  - The library is released as a single JavaScript file
    - Can be downloaded then installed locally
    - Include it from a CDN like Google, Microsoft or jQuery itself
- Advantage of using CDN host:
  - The bandwidth of the file is offloaded to reduce the demand on your servers.
  - The user may already have cached the third-party file and thus not have to download it again, thereby reducing the total loading time.
- A disadvantage to the third-party CDN is that your jQuery will fail if the third-party host fails (unlikely but possible)

Loading jQuery

```
<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
<script type="text/javascript">
window.jQuery ||
document.write('<script src="/jquery-1.9.1.min.js"><\/script>');
</script>
```

LISTING 15.5 jQuery loading using a CDN and a local fail-safe if the CDN is offline

- The basic syntax is: **`$(selector).action()`**
  - A **`$`** sign to define/access jQuery
  - A **`(selector)`** to "query (or find)" HTML elements
  - A jQuery **`action()`** to be performed on the element(s)

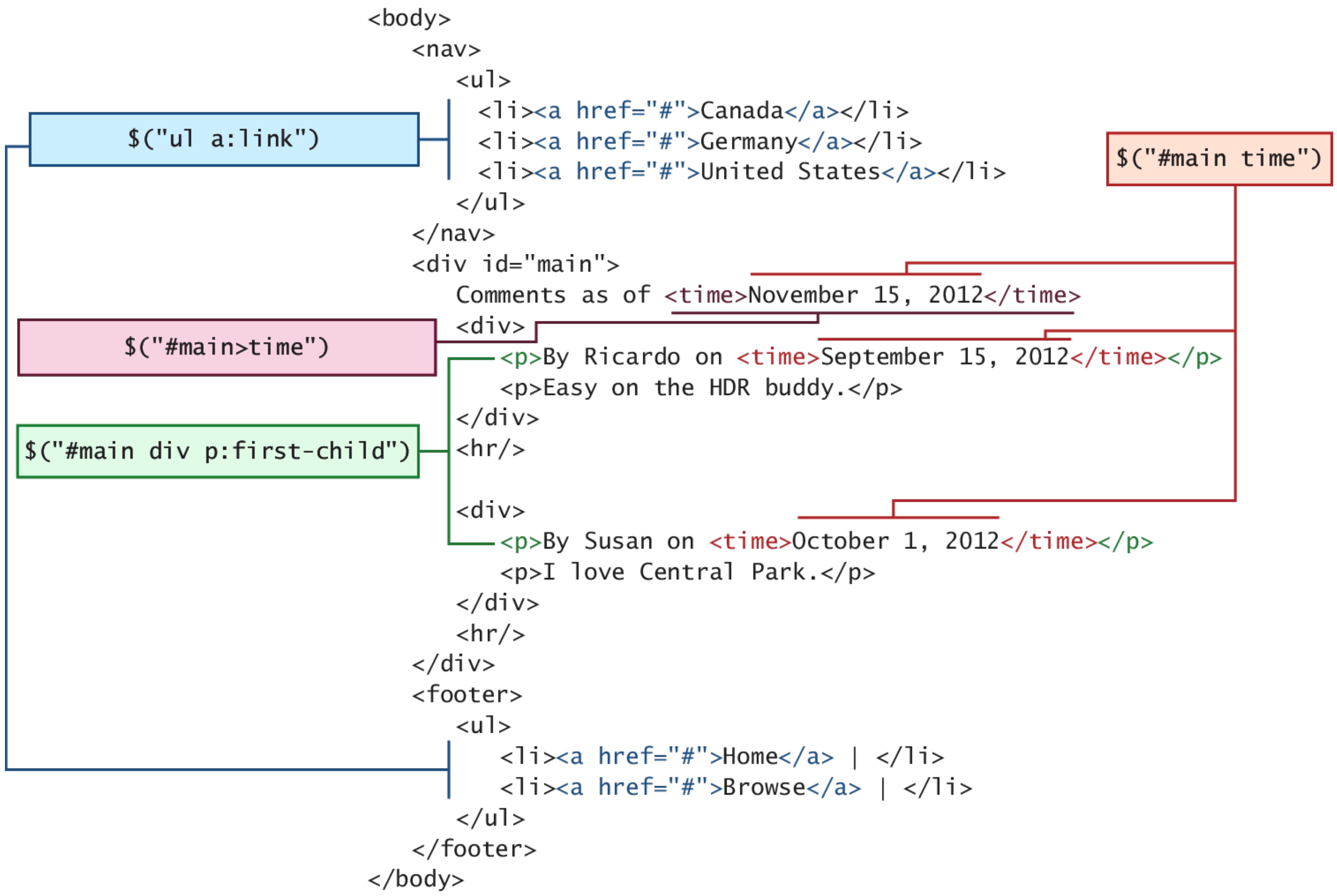
jQuery selectors

- The selectors are very similar to CSS selectors
- The four basic selectors are:
  - $("*") **Universal selector** matches all elements (and is slow).
  - $("tag") **Element selector** matches all elements with the given element name.
  - $(".class") **Class selector** matches all elements with the given CSS class.
  - $("#id") **Id selector** matches all elements with a given HTML id attribute.
- Other selectors defined in CSS can be used

Basic Selector example

- For example, to select the single <div> element with id="grab" you would write:
  - **var singleElement = $("#grab");**
- To get a set of all the <a> elements the selector would be:
  - **var allAs = $("a");**
- These selectors replace the use of **`getElementById()`** and similar functions entirely.

CSS Selectors Review



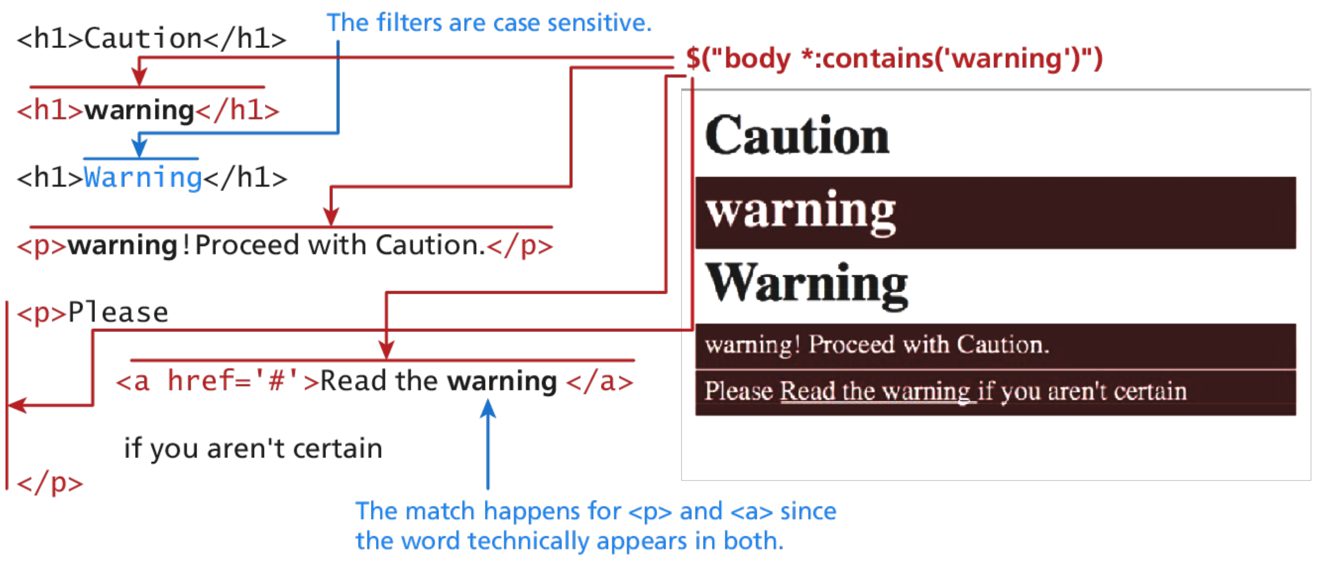
```
<body>
   <nav>
      <ul>
        <li><a href="#">Canada</a></li>
        <li><a href="#">Germany</a></li>
        <li><a href="#">United States</a></li>
      </ul>
   </nav>
   <div id="main">
      Comments as of <time>November 15, 2012</time>
      <div>
         <p>By Ricardo on <time>September 15, 2012</time></p>
         <p>Easy on the HDR buddy.</p>
      </div>
      <hr/>

      <div>
         <p>By Susan on <time>October 1, 2012</time></p>
         <p>I love Central Park.</p>
      </div>
      <hr/>
   </div>
   <footer>
      <ul>
         <li><a href="#">Home</a> | </li>
         <li><a href="#">Browse</a> | </li>
      </ul>
   </footer>
</body>
```

More selectors

- Pseudo class selector
  - E.g. Selecting all links that have been visited
  - **var visitedLinks = $("a:visited");**
- Beyond CSS selectors
  - Content Filters
    - Select elements based on criteria like if the element has a particular child, or no children or contains certain piece of text
  - Form Selectors
    - Shorthand version to select form elements

Content Filters

- **$("body *:contains('warning')")**


```
<h1>Caution</h1>
<h1>warning</h1>
<h1>Warning</h1>
<p>warning!Proceed with Caution.</p>
<p>Please
   <a href='#'>Read the warning </a>
   if you aren't certain
</p>
```
The filters are case sensitive.
$("body *:contains('warning')")
The match happens for <p> and <a> since the word technically appears in both.

Caution

warning

Warning

warning! Proceed with Caution.

Please Read the warning if you aren't certain

Form Selectors

Selector	CSS Equivalent	Description
$(:button)	$("button, input[type='button']")	Selects all buttons
$(:checkbox)	$('[type=checkbox]‘)	Selects all checkboxes
$(:checked)	No Equivalent	Selects elements that are checked. This includes radio buttons and checkboxes.
$(:disabled)	No Equivalent	Selects form elements that are disabled.
$(:enabled)	No Equivalent	Opposite of :disabled
$(:file)	$('[type=file]')	Selects all elements of type file
$(:focus)	$(document.activeElement)	The element with focus
$(:image)	$('[type=image]')	Selects all elements of type image
$(:input)	No Equivalent	Selects all <input>, <textarea>, <select>, and <button> elements.
$(:password)	$('[type=password]')	Selects all password fields
$(:radio)	$('[type=radio]')	Selects all radio elements
$(:reset)	$('[type=reset]')	Selects all the reset buttons
$(:selected)	No Equivalent	Selects all the elements that are currently selected of type <option>. It does not include checkboxes or radio buttons.
$(:submit)	$('[type=submit]')	Selects all submit input elements
$(:text)	No Equivalent	Selects all input elements of type text. $('[type=text]') is almost the same, except that $(:text) includes <input> fields with no type specified.

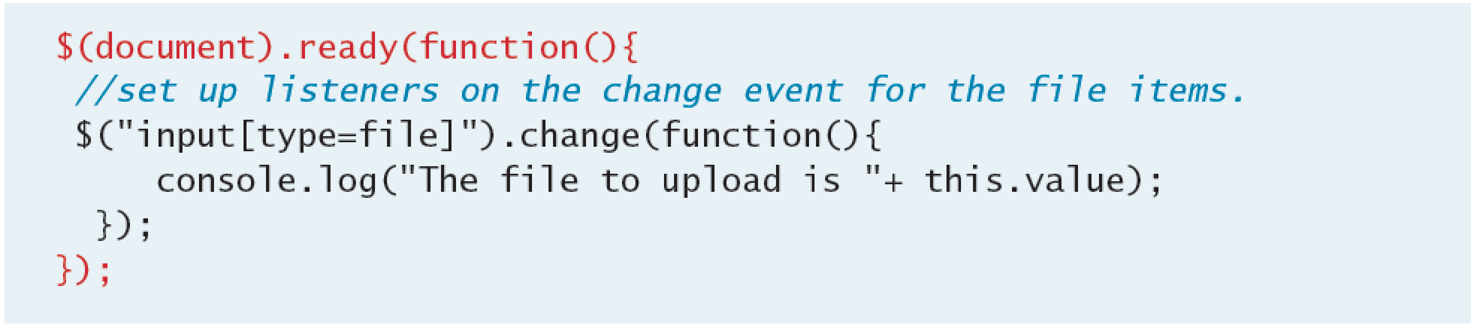
Outline

- Intro to jQuery
  - Selectors
  - Event handler and DOM man
  - Ajax requests
- Integrate jQuery with Expressjs Application

Registering event handler

- Standard even handling syntax

  -
    ```
    $("p").click(function(){
      // action goes here!!
    });
    ```
- Common DOM events are
  - `click, dbclick, mouseenter, mouseleave`
- The `Document Ready` Event
  - It is good practice to put jQuery code inside a document ready event
  - The ready event is defined by jQuery, fired after the DOM is completed



```
$(document).ready(function(){
 //set up listeners on the change event for the file items.
 $("input[type=file]").change(function(){
     console.log("The file to upload is "+ this.value);
  });
});
```

LISTING 15.6 jQuery code to listen for file inputs changing, all inside the document's ready event

Normal DOM manipulation

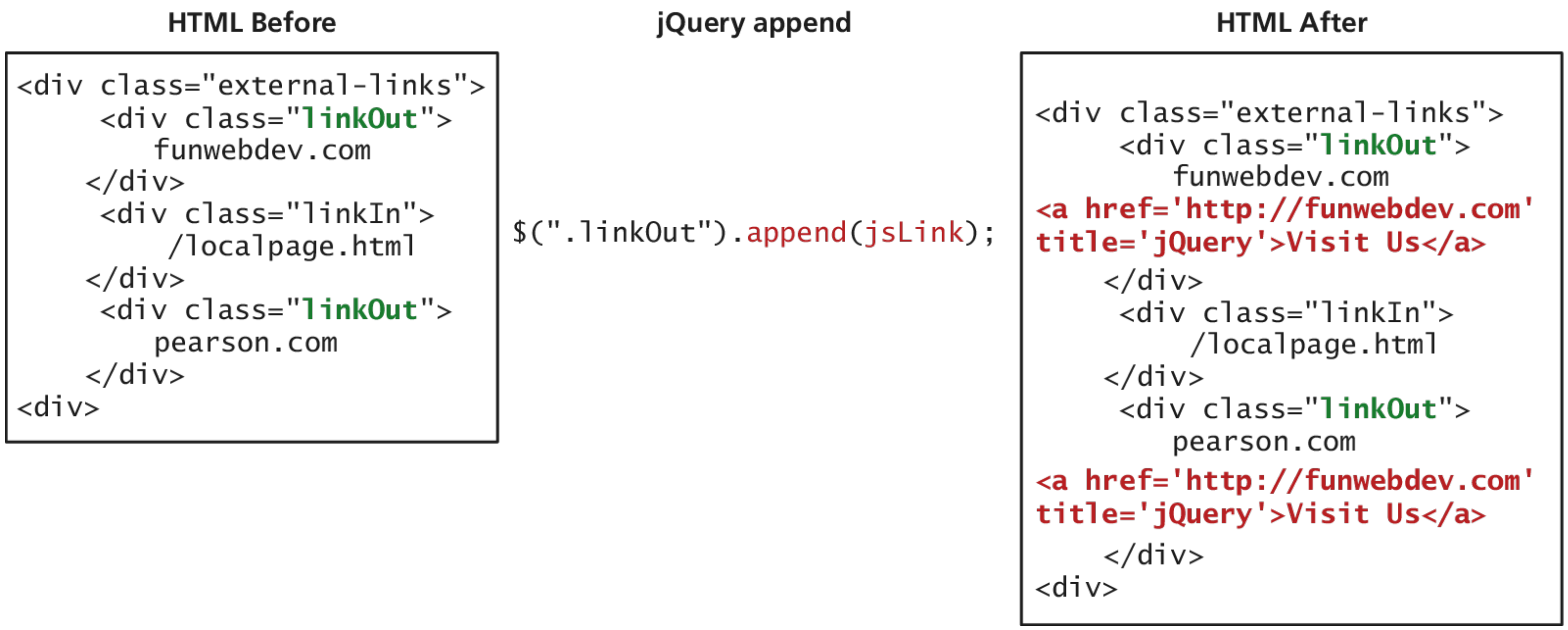
Creating Element

```
 // pure JavaScript way
var jsLink = document.createElement("a");
jsLink.href = "http://www.funwebdev.com";
jsLink.innerHTML = "Visit Us";
jsLink.title = "JS";

// jQuery way
var jQueryLink = $("<a href='http://funwebdev.com'
title = 'jQuery'>Visit Us</a>");

// jQuery long-form way
var jQueryVerboseLink = $("<a></a>");
jQueryVerboseLink.attr("href",'http://funwebdev.com');
jQueryVerboseLink.attr("title","jQuery verbose");
jQueryVerboseLink.html("Visit Us");
```

LISTING 15.8 A comparison of node creation in JS and jQuery

Normal DOM manipulation

- Appending DOM Elements
  - The append() method takes as a parameter an HTML string, a DOM object, or a jQuery object. That object is then added as the last child to the element(s) being selected.



HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

jQuery append

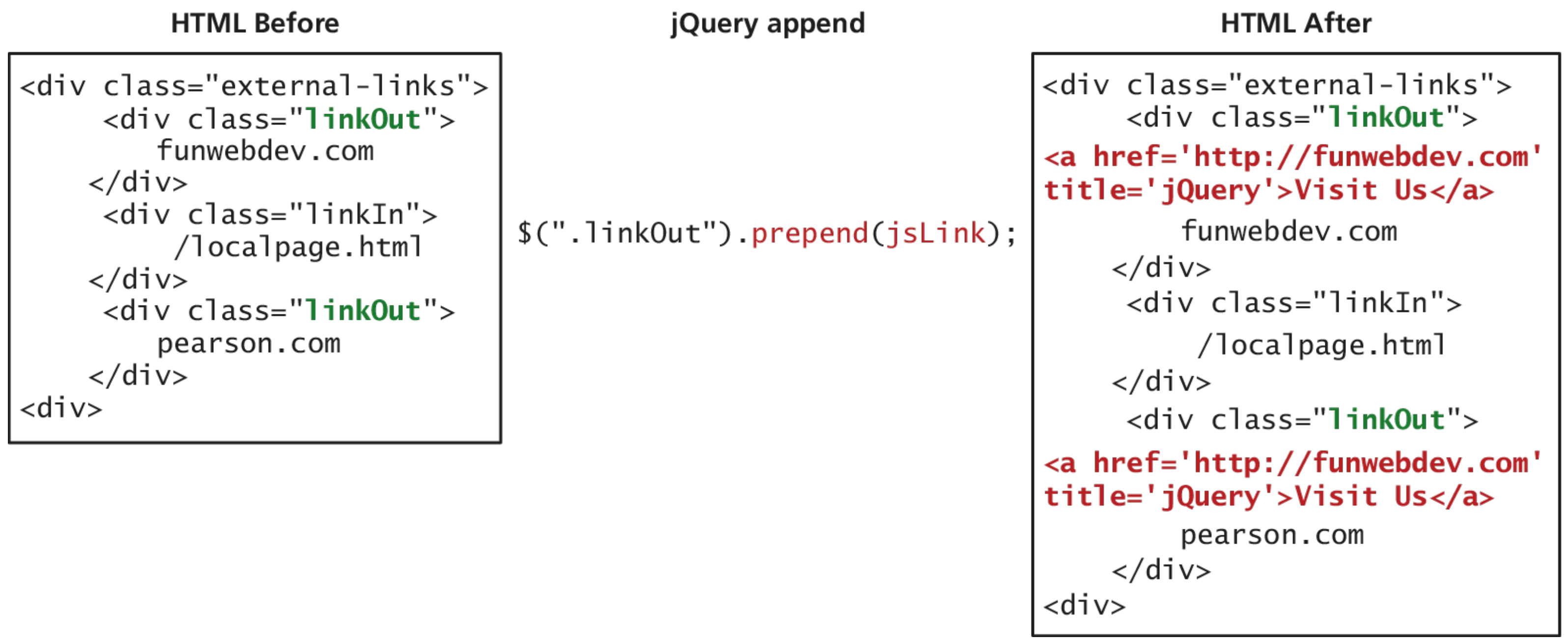
```
$(".linkOut").append(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
    </div>
<div>
```

Normal DOM manipulation

- Prepending DOM Elements
  - The prepend() and prependTo() methods operate in a similar manner except that they add the new element as the first child rather than the last.



HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

jQuery append

```
$(".linkOut").prepend(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
        pearson.com
    </div>
<div>
```

Useful methods

- We can both set and get an attribute value by using the **attr()** method on any element from a selector.
  - **var link = $("a").attr("href");**
  - **$("img").attr("class","fancy");**
- CSS properties can be set and get with css() method on any element from a selector
  - $("#colourBox").**css**("**background-color**", "**#FF0000**")
- The **html()** method is used to get the HTML contents of an element. If passed with a parameter, it updates the HTML of that element.
- The **val()** method returns the value of the element. It is mainly used to get the value of form element.

Outline

- Intro to jQuery
  - Selectors
  - Event handler and DOM man
  - Ajax requests
- Integrate jQuery with Expressjs Application

Synchronous and Asynchronous Request

- Traditional Web application use only **synchronous** request
  - Browser send a request then **WAIT** for the response and render the page based on the response
  - While a **synchronous** request is being processed on the server, the user cannot interact with the client web browser
  - The "responsiveness" of the traditional web application is much worse than that of a desktop application
  - The synchronous model was originally designed for a web of hypertext documents
- **Asynchronous** request
  - **Asynchronous** request allow the user to continue interacting with the application while the server processes the request concurrently
  - This is achieved by client side script (usually JavaScript) creating an **XMLHttpRequest** object to manage a request and use implicit or explicit callback function to handle the response.

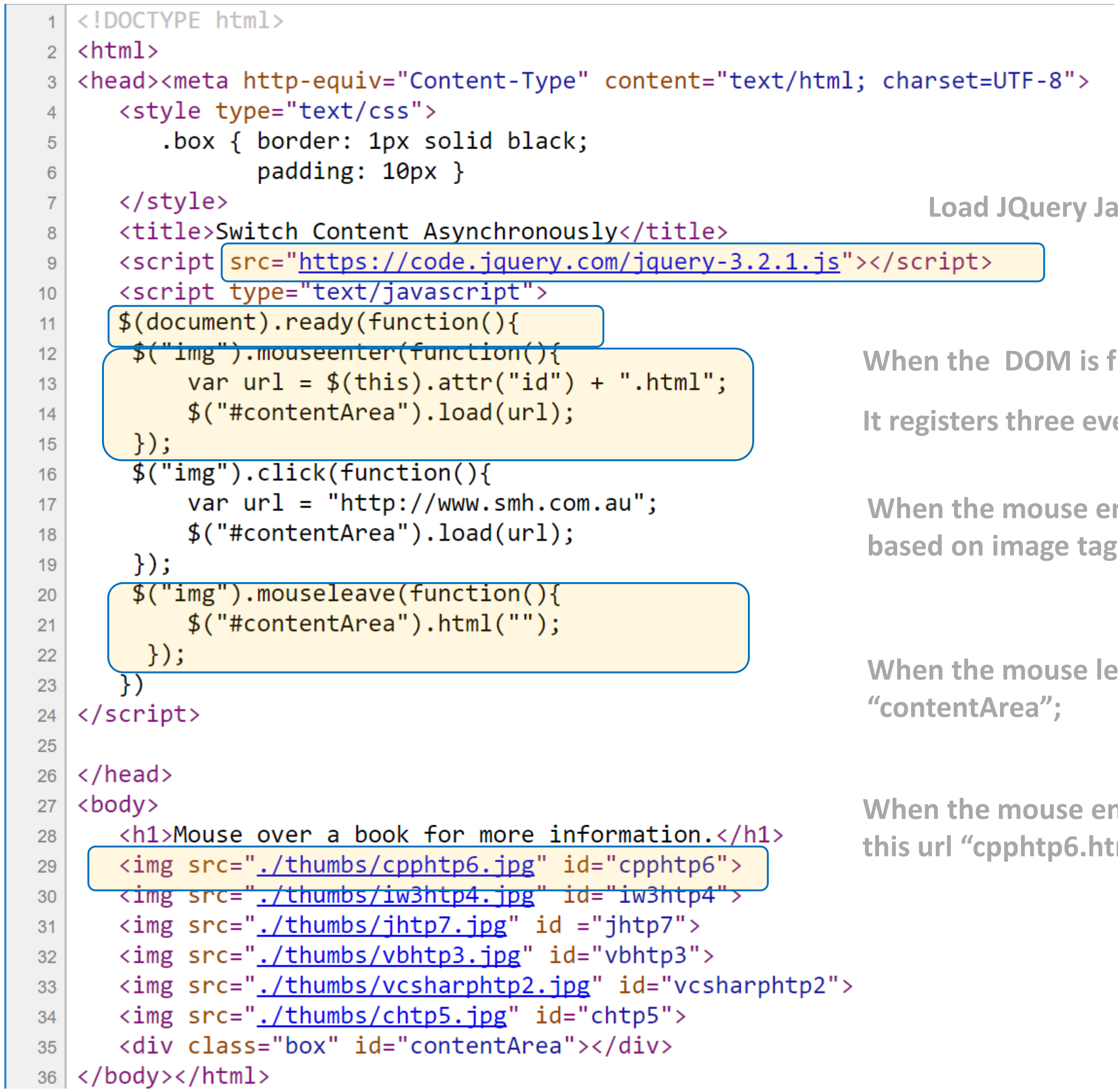
A simple example

- The XMLHttpRequest object will fetch a static file from the server, the JavaScript running on the client browser dynamically insert the content into the current DOM tree.

http://web.it.usyd.edu.au/~comp5347/2017/week10/SwitchContent.html

http://web.it.usyd.edu.au/~comp5347/2017/week10/SwitchContentJQuery.html

Source code of the simple example



```
<!DOCTYPE html>
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <style type="text/css">
      .box { border: 1px solid black;
             padding: 10px }
   </style>
   <title>Switch Content Asynchronously</title>
   <script src="https://code.jquery.com/jquery-3.2.1.js"></script>
   <script type="text/javascript">
   $(document).ready(function(){
    $("img").mouseenter(function(){
        var url = $(this).attr("id") + ".html";
        $("#contentArea").load(url);
    });
    $("img").click(function(){
        var url = "http://www.smh.com.au";
        $("#contentArea").load(url);
    });
    $("img").mouseleave(function(){
        $("#contentArea").html("");
     });
   })
</script>

</head>
<body>
   <h1>Mouse over a book for more information.</h1>
   <img src="./thumbs/cpphtp6.jpg" id="cpphtp6">
   <img src="./thumbs/iw3htp4.jpg" id="iw3htp4">
   <img src="./thumbs/jhtp7.jpg" id ="jhtp7">
   <img src="./thumbs/vbhtp3.jpg" id="vbhtp3">
   <img src="./thumbs/vcsharphtp2.jpg" id="vcsharphtp2">
   <img src="./thumbs/chtp5.jpg" id="chtp5">
   <div class="box" id="contentArea"></div>
</body></html>
```

Load JQuery JavaScript library

When the DOM is fully loaded, execute this function;

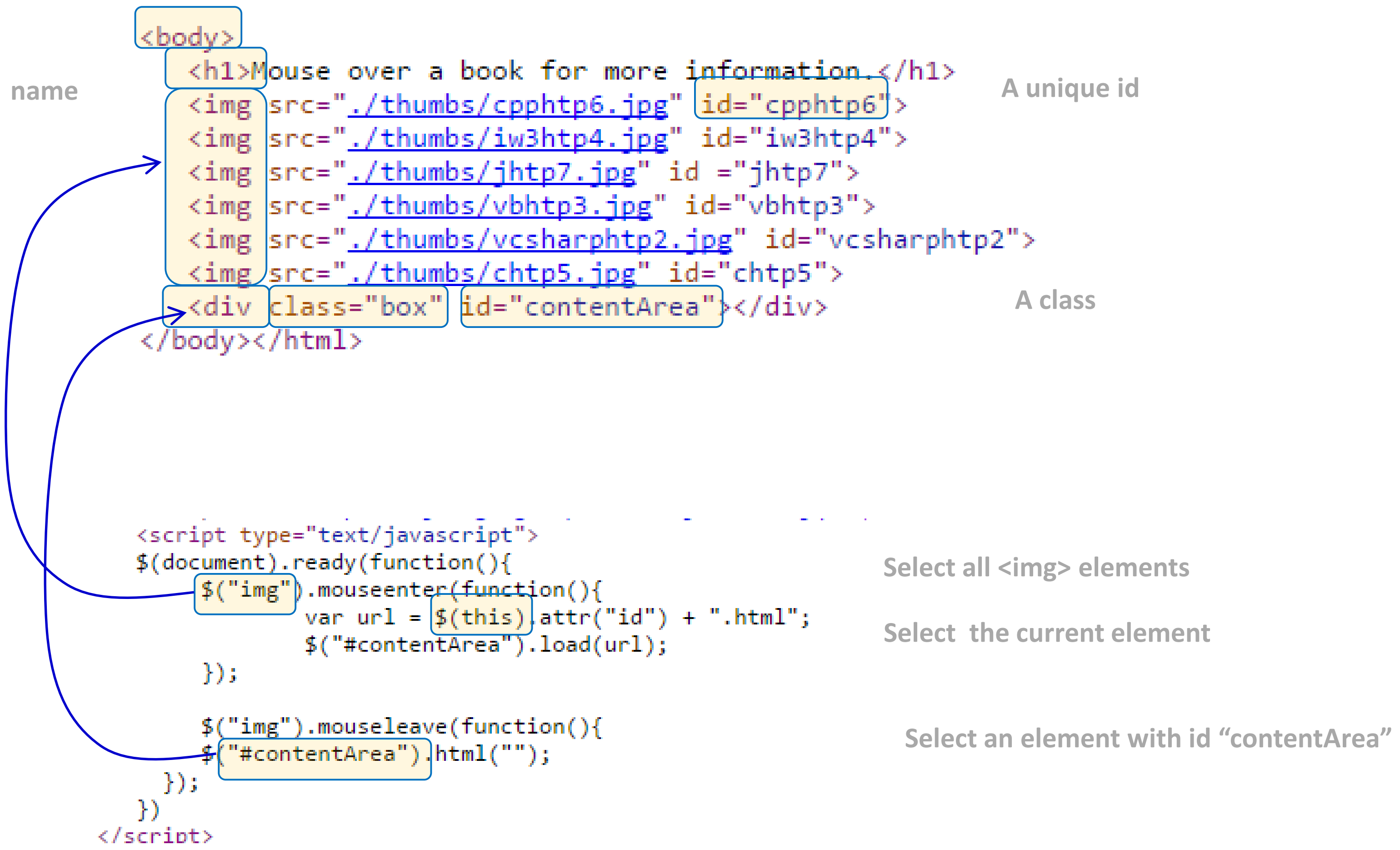
It registers three event handler functions to the <img> tag ;

When the mouse enters an imageThe url is constructed based on image tag's id value

When the mouse leaves an image, clear the tag with id "contentArea";

When the mouse enters this image, load the content form this url "cpphtp6.html" to the division with id "contentArea"

Selectors in the Example code



```
<body>
  <h1>Mouse over a book for more information.</h1>
  <img src="./thumbs/cpphtp6.jpg" id="cpphtp6">
  <img src="./thumbs/iw3htp4.jpg" id="iw3htp4">
  <img src="./thumbs/jhtp7.jpg" id ="jhtp7">
  <img src="./thumbs/vbhtp3.jpg" id="vbhtp3">
  <img src="./thumbs/vcsharphtp2.jpg" id="vcsharphtp2">
  <img src="./thumbs/chtp5.jpg" id="chtp5">
  <div class="box" id="contentArea"></div>
</body></html>
```

```
<script type="text/javascript">
$(document).ready(function(){
    $("img").mouseenter(function(){
            var url = $(this).attr("id") + ".html";
            $("#contentArea").load(url);
    });

    $("img").mouseleave(function(){
    $("#contentArea").html("");
   });
  })
</script>
```

name
A unique id
A class
Select all <img> elements
Select the current element
Select an element with id “contentArea”

jQuery Ajax support

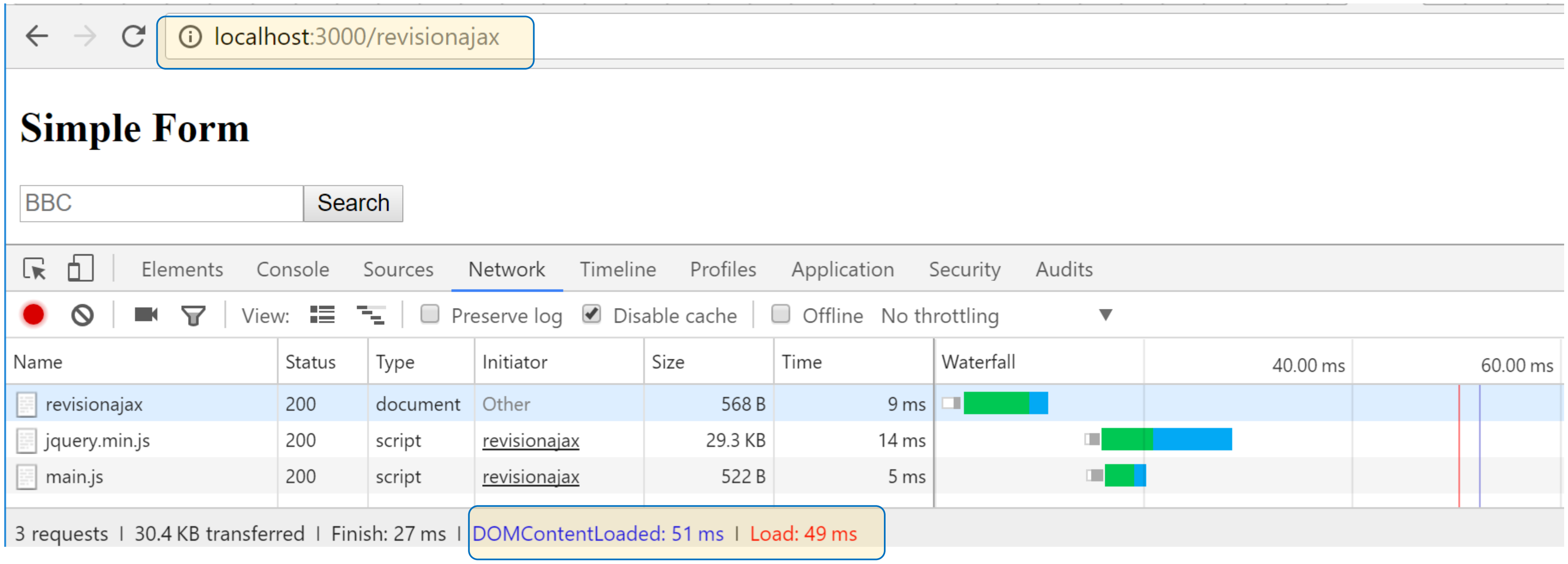
- jQuery provides many methods to support asynchronous requests. Below are a few common ones
- **`load()`**: **`$(selector).load(URL,data,callback);`**
  - Load URL’s response into the selected element, optional `data` can be sent along with the request; optional callback can be executed after load() finishes
  - A GET request is sent if no data is present, otherwise a POST request is sent
- **`get()`: `$.get(URL,callback);`**
  - Request data using HTTP GET method; the optional callback parameter is the name of a function to be after the response arrives
- **`post():$.getJSON(URL,data, callback);`**
  - Request data using HTTP POST methods; optional `data` can be sent along with the request; optional callback can be executed after response arrives

Outline

- Intro to jQuery
  - Selectors
  - Event handler and DOM man
  - Ajax requests
- Integrate jQuery with Expressjs Application

Express with jQuery powered Ajax frontend

- We can add ajax support to the simple app in week 9



localhost:3000/revisionajax

Simple Form

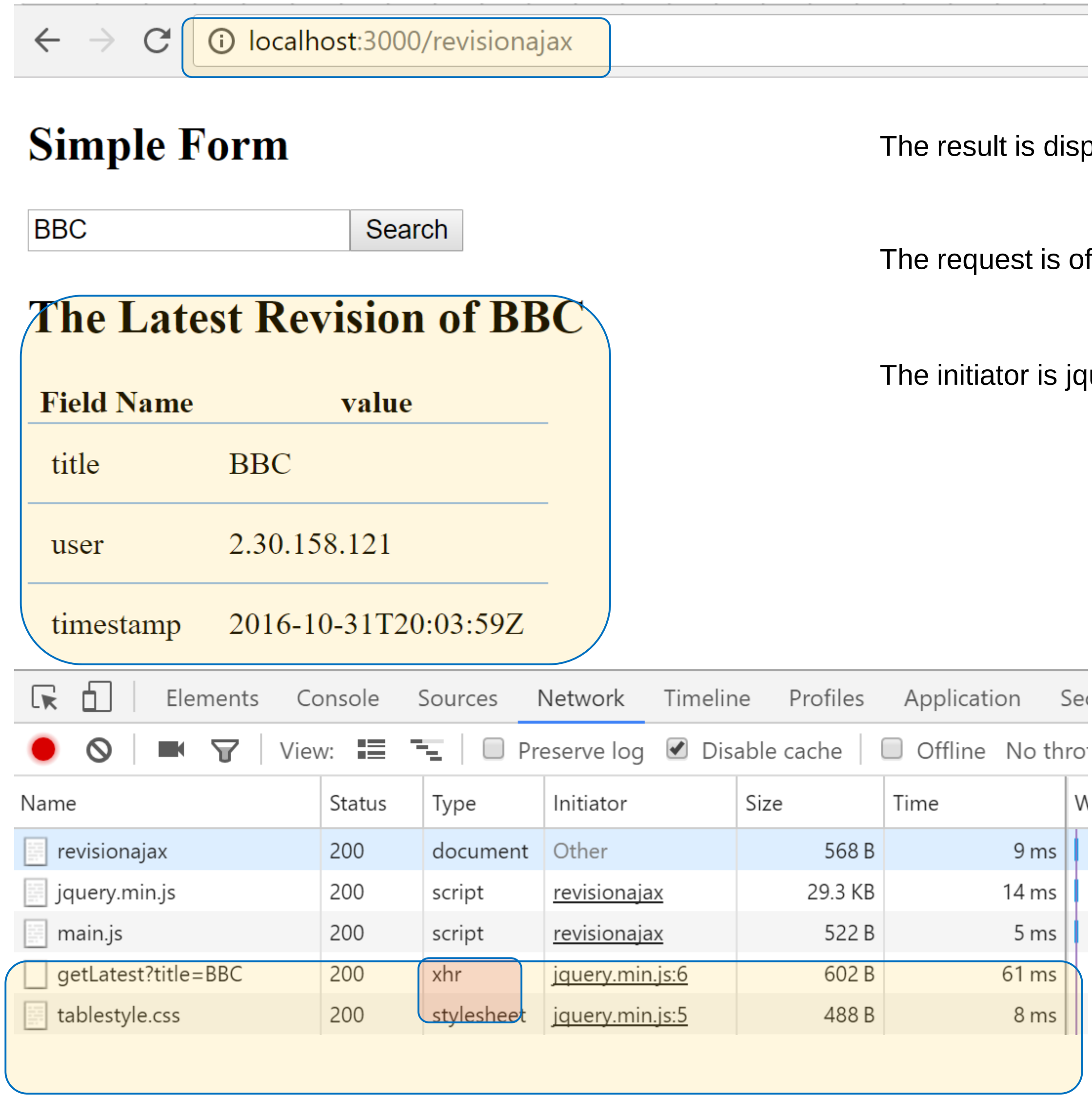
BBC | Search

Name	Status	Type	Initiator	Size	Time
revisionajax	200	document	Other	568 B	9 ms
jquery.min.js	200	script	revisionajax	29.3 KB	14 ms
main.js	200	script	revisionajax	522 B	5 ms

3 requests | 30.4 KB transferred | Finish: 27 ms | DOMContentLoaded: 51 ms | Load: 49 ms

Effect of render blocking JavaScript

Ajax frontend output



localhost:3000/revisionajax

Simple Form

BBC | Search

The Latest Revision of BBC

Field Name	value
title	BBC
user	2.30.158.121
timestamp	2016-10-31T20:03:59Z

Name	Status	Type	Initiator	Size	Time
revisionajax	200	document	Other	568 B	9 ms
jquery.min.js	200	script	revisionajax	29.3 KB	14 ms
main.js	200	script	revisionajax	522 B	5 ms
getLatest?title=BBC	200	xhr	jquery.min.js:6	602 B	61 ms
tablestyle.css	200	stylesheet	jquery.min.js:5	488 B	8 ms

The result is displayed on the same page

The request is of type xhr

The initiator is jquery.min.js

Changes to the title form view

- Add a place holder for results
- Add reference to scripts
- Change the submit button behaviour

titleFormAjax.pug

```
doctype html

html(lang="en")
head
  title Ajax Search Example
    script(src="https://code.jquery.com/jquery-3.2.1.js")
    script(src="/js/main.js")
body
  h2 Simple Form
  input#title(type="search", placeholder="BBC")
  button#button(type='button') Search
  div#results
```

The client side script

```
$(document).ready(function(){
  $('#button').on('click', function(e){
      var parameters = {title: $('#title').val() };
    $.get( 'revisionajax/getLatest',parameters, function(result) {
      $('#results').html(result);
    });
  });
});
```

These two are equivalent. The top one use `$.get(url, data,callback)`,
The bottom one uses `element.load(url)`

```
$(document).ready(function(){
  $('#button').on('click', function(e){
    var data=$('#title').val();
    $('#results').load('revisionajax/getLatest?title='+data)
  });
});
```

What else do we need to change?

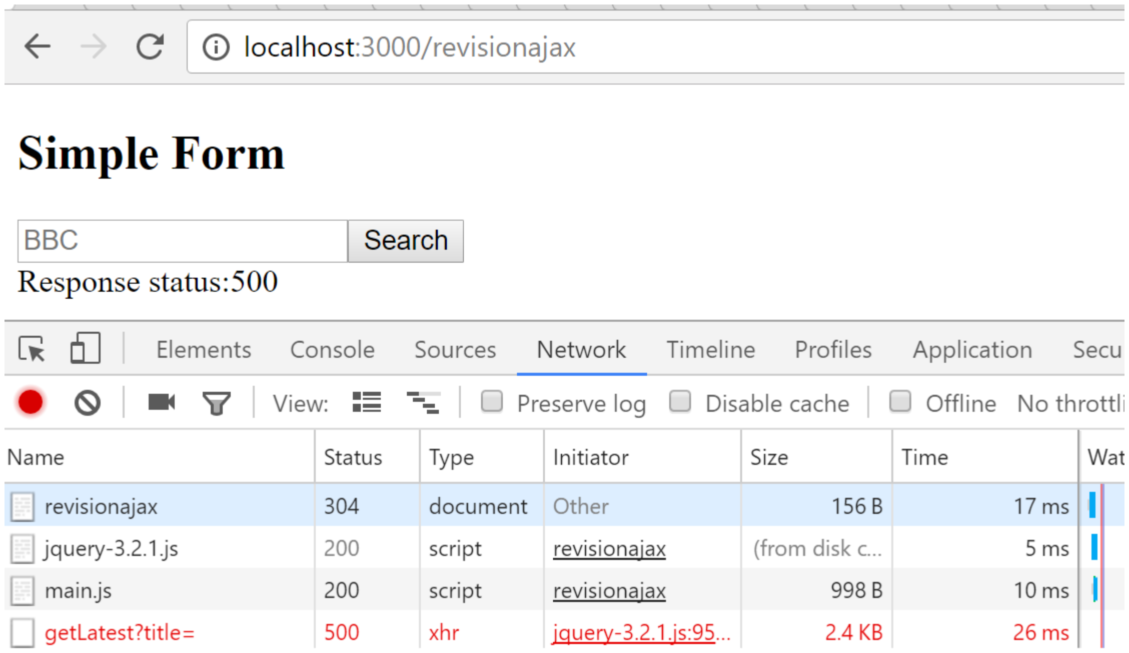
- No major change except a few “wirings”
- The url to controller mapping
- Controller to new view

The jqXHR Object

- All jQuery Ajax requests return a jqXHR object to encapsulate the response from the server. The object is a superset of the original XMLHTTPRequest object
- This object can be used to write to handle various server response: success, failure
  - We assume the request is always successful previously
- The jqXHR.done() (for success), jqXHR.fail() (for error), and jqXHR.always() is like the regular try(){}catch(){}finally() block
-

The jqXHR Object Example

```
$(document).ready(function(){
  $('#button').click(function(e){
var parameters = {title: $('#title').val() };
      var jqxhr = $.get( 'revisionajax/getLatest',parameters)
      jqxhr.done(function(result) {
         $('#results').html(result);
      });
      jqxhr.fail(function(jqXHR){
         $('#results').html("Response status:" + jqXHR.status)
      //console.log("Response status:" + jqXHR.status)
      })
  });
});
```



localhost:3000/revisionajax

Simple Form

BBC | Search

Response status:500

Name	Status	Type	Initiator	Size	Time	Wat
revisionajax	304	document	Other	156 B	17 ms	
jquery-3.2.1.js	200	script	revisionajax	(from disk c...	5 ms	
main.js	200	script	revisionajax	998 B	10 ms	
getLatest?title=	500	xhr	jquery-3.2.1.js:95...	2.4 KB	26 ms	

http://api.jquery.com/jQuery.ajax/#jqXHR

Same Origin Policy

- The most important security concept in modern browser
  - Mostly, restrict what resources JavaScript (and other scripting language) can access inside a browser
    - DOM, Cookie, XMLHttpRequest, and so on
- An origin is defined by
  - Protocol
  - Host name
  - Port number
- If two pages are from same origin, the web browser permits scripts from one page to access data in a second page.

Ajax: Same Origin Policy

- XMLHttpRequest Security
  - `XMLHttpRequest` object does not allow a web application to request resources from servers other than the one that served the web application (SOP on XHR)
  - There are various ways to circumvent this security restriction
  - You can implement a server-side proxy—an application on the web application's web server—that can make requests to other servers on the web application's behalf
  - **Cross-origin resource sharing (CORS)** uses new headers in the HTML5 standard to let site specify other domains that can share its content through JavaScript

Access-Control-Allow-Origin: www.funwebdev.com

Admin

- There will be a quiz next week (week 11)
  - Duration: 1 hour
  - Format: paper based (MCQ and short answer questions)
  - Content: week 1~ week 10
  - Time: Tuesday evening (23rd of May) **7:30pm – 8:30pm**
  - Location
    - If you are in Tuesday evening lab: SIT114, SIT115, SIT130B and SIT 457, go to your allocated lab room
    - If you are in Tuesday evening lab 116 and Wednesday lab, stay in the lecture room for your quiz

Resources

- W3C school jQuery Tutorial
  - http://www.w3schools.com/jquery/default.asp
- Fundamentals of Web Development
  - Chapter 15