Chapter 6
DOM – AJAX - jQuery

Lectured by:
Nguyễn Hữu Hiếu

DOM
Document Object Model

DOM & DHTML

- Dynamic web pages with JavaScript and DOM
  - DHTML (Dynamic HTML)
- DOM nodes and DOM tree
- Traversing, editing and modifying DOM nodes
- Editing text nodes
- Accessing, editing and modifying elements' attributes

DOM Concept

- DOM makes all components of a web page accessible
  - HTML elements
  - their attributes
  - text
- They can be created, modified and removed with JavaScript

DOM Objects

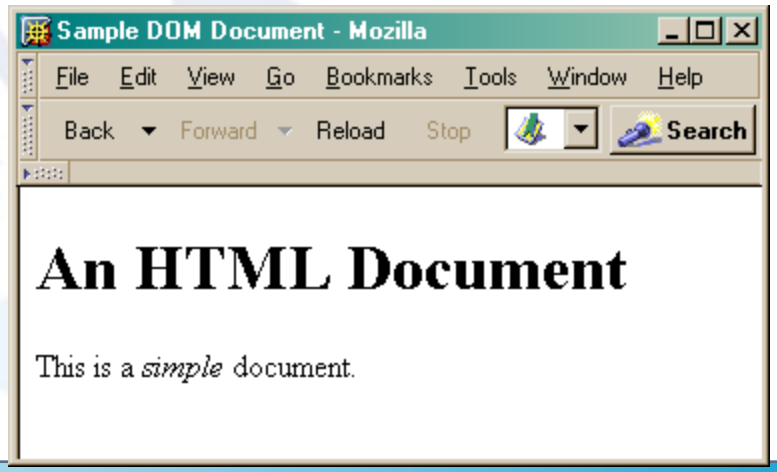
- DOM components are accessible as objects or collections of objects
- DOM components form a tree of nodes
  - relationship parent node – children nodes
  - `document` is the root node
- Attributes of elements are accessible as text
- Browsers can show DOM visually as an expandable tree
  - Firebug for Firefox
  - in IE -> Tools -> Developer Tools

Example

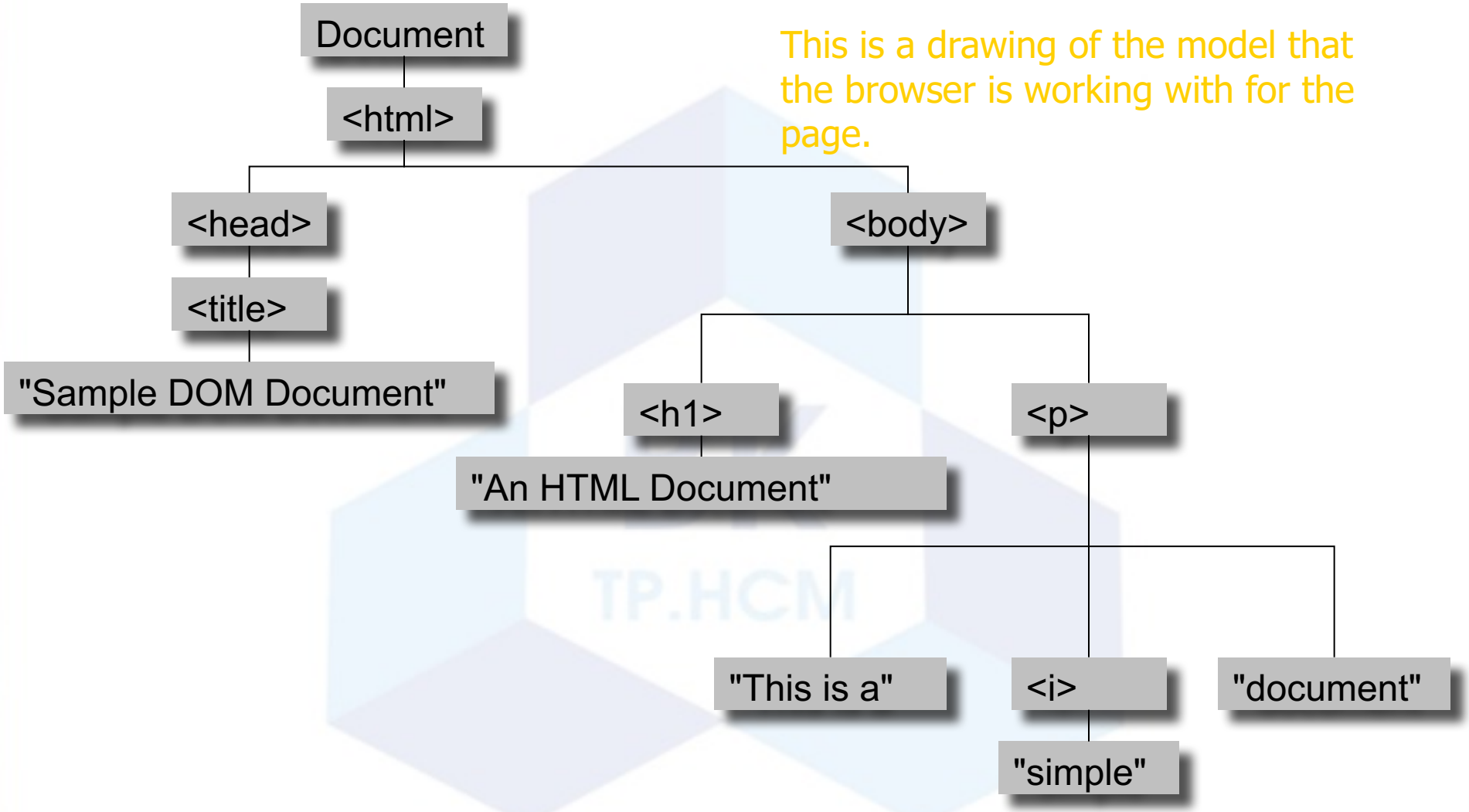
This is what the browser reads

```
<html>
  <head>
    <title>Sample DOM Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
  </body>
</html>
```

This is what the browser displays on screen.



Example



This is a drawing of the model that the browser is working with for the page.

DOM Standards

- W3C www.w3.org defines the standards
- DOM Level 3 recommendation
  - www.w3.org/TR/DOM-Level-3-Core/
- DOM Level 2 HTML Specification
  - www.w3.org/TR/DOM-Level-2-HTML/
  - additional DOM functionality specific to HTML, in particular objects for XHTML elements
- **But,** the developers of web browsers
  - **don't** implement all standards
  - implement some standards **differently**
  - implement some **additional features**

Accessing Nodes by `id`

- Access to elements by their `id`
  - `document.getElementById(<id>)`
    - returns the element with `id` `<id>`
  - `id` attribute can be defined in each start tag
    - `div` element with `id` attribute can be used as an root node for a dynamic DOM subtree
    - `span` element with `id` attribute can be used as a dynamic inline element
- The preferred way to access elements

Other Access Methods

- Access by elements' tag
  - there are typically several elements with the same tag
  - `document.getElementsByTagName(<tag>)`
    - returns the collection of all elements whose tag is `<tag>`
    - the collection has a `length` attribute
    - an item in the collection can be reached by its index
  - e.g.
    - `var html = document.getElementsByTagName("html")[0];`
- Access by elements' `name` attribute
  - several elements can have the same name
  - `document.getElementsByName(<name>)`
    - returns the collection of elements with `name` `<name>`

Traversing DOM tree

- Traversal through node properties
  - `childNodes` property
    - the value is a collection of nodes
      - has a `length` attribute
      - an item can be reached by its index
    - e.g. `var body = html.childNodes[1];`
  - `firstChild`, `lastChild` properties
  - `nextSibling`, `previousSibling` properties
  - `parentNode` property

Other Node Properties

- `nodeType` property
  - `ELEMENT_NODE`: HTML element
  - `TEXT_NODE`: text within a parent element
  - `ATTRIBUTE_NODE`: an attribute of a parent element
    - attributes can be accessed another way
  - `CDATA_SECTION_NODE`
    - CDATA sections are good for unformatted text
- `nodeName` property
- `nodeValue` property
- `attributes` property
- `innerHTML` property
  - not standard, but implemented in major browsers
  - very useful
- `style` property
  - object whose properties are all style attributes, e.g., those defied in CSS

Accessing JS Object's Properties

- There are two different syntax forms to access object's properties in JS (
  - `<object>.<property>`
    - dot notation, e.g., `document.nodeType`
  - `<object>[<property-name> ]`
    - brackets notation, e.g., `document["nodeType"]`
    - this is used in `for-in` loops

- this works for properties of DOM objects, too

Attributes of Elements

- Access through `attributes` property
  - `attributes` is an array
  - has a `length` attribute
  - an item can be reached by its index
  - an item has the properties `name` and `value`
  - e.g.
    - `var src = document.images[0].attributes[0].value;`
- Access through function `getAttribute(<name>)`
  - returns the value of attribute `<name>`
  - e.g.
    - `var src = document.images[0].getAttribute("src");`

Text Nodes

- Text node
  - can only be as a leaf in DOM tree
  - it's `nodeValue` property holds the text
  - `innerHTML` can be used to access the text

- Watch out:
  - **There are many more text nodes than you would expect!**

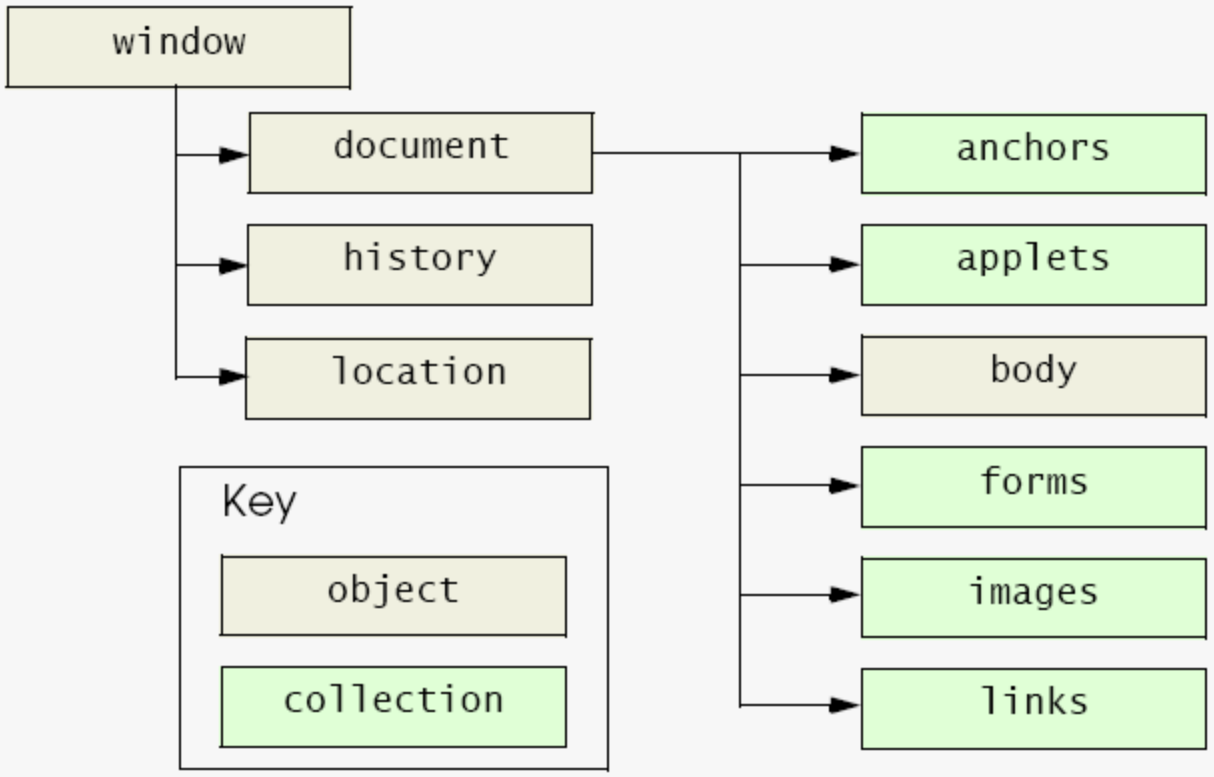
Modifying DOM Structure

- **document.createElement(<tag>)**
  - creates a new DOM element node, with **<tag>** tag.
  - the node still needs to be inserted into the DOM tree
- **document.createTextNode(<text>)**
  - creates a new DOM text with **<text>**
  - the node still needs to be inserted into the DOM tree
- **<parent>.appendChild(<child>)**
  - inserts <child> node behind all existing children of <parent> node
- **<parent>.insertBefore(<child>,<before>)**
  - inserts **<child>** node before **<before>** child within <parent> node
- **<parent>.replaceChild(<child>,<instead>)**
  - replaces **<instead>** child by **<child>** node within <parent> node
- **<parent>.removeChild(<child>)**
  - removes **<child>** node from within <parent> node

Modifying Node Attributes

- **<node>.setAttribute(<name>,<value>)**
  - sets the value of attribute **<name> to <value>**
  - e.g.
    - `document.images[0].setAttribute("src","keiki.jpg");`

- That's the standard
  - but it doesn't work in IE, there you have to use
    - **setAttribute(<name=value>)**
  - e.g.
    - `document.images[0].setAttribute("src=\"keiki.jpg\"");`


window
document
history
location
anchors
applets
body
forms
images
links
Key
object
collection

Special DOM Objects

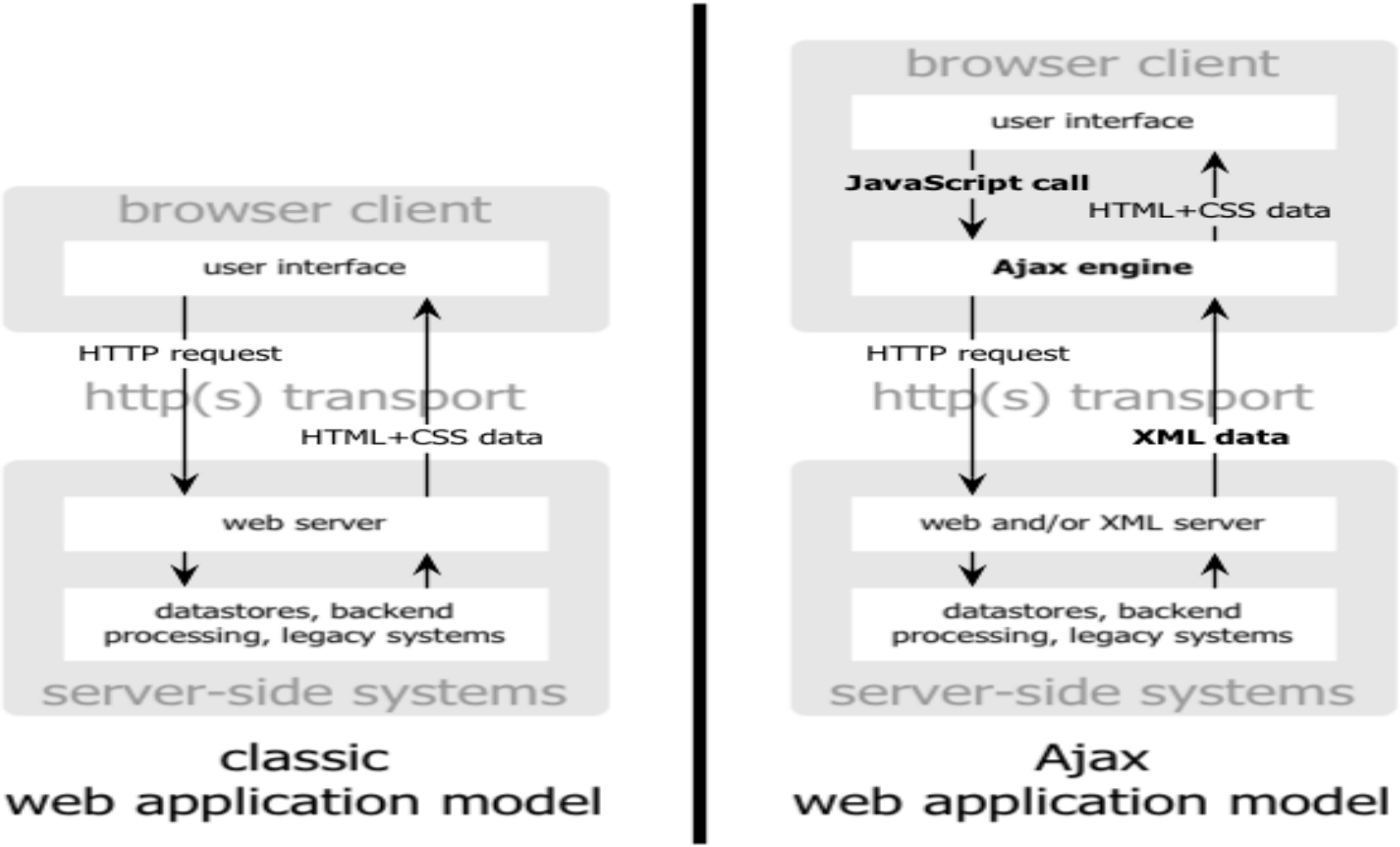
- `window`
  - the browser window
  - new popup `window`s can be opened
- `document`
  - the current web page inside the `window`
- `body`
  - `<body>` element of the `document`
- `history`
  - sites that the user visited
  - makes it possible to go back and forth using scripts
- `location`
  - URL of the `document`
  - setting it goes to another page

AJAX
Asynchronous JavaScript And XML

AJAX

- A lot of hype
  - It has been around for a while
  - Not complex
- Powerful approach to building websites
  - Think differently
- Allows for more interactive web applications
  - Gmail, docs.google.com, Flickr, ajax13, etc.

AJAX


browser client
user interface
HTTP request
http(s) transport
HTML+CSS data
web server
datastores, backend processing, legacy systems
server-side systems
classic web application model
browser client
user interface
JavaScript call
HTML+CSS data
Ajax engine
HTTP request
http(s) transport
XML data
web and/or XML server
datastores, backend processing, legacy systems
server-side systems
Ajax web application model

AJAX Technologies

- HTML
  - Used to build web forms and identify fields
- Javascript
  - Facilitates asynchronous communication and modification of HTML in-place
- DHTML - Dynamic HTML
  - Additional markup for modifying and updating HTML
- DOM - Document Object Model
  - Used via Javascript to work with both the structure of your HTML and also XML from the server

The XMLHttpRequest Object

- Base object for AJAX
  - Used to make connections, send data, receive data, etc.
- Allows your javascript code to talk back and forth with the server all it wants to, without the user really knowing what is going on.
- Available in most browsers
  - But called different things

The XMLHttpRequest Object

```
 <script language="javascript" type="text/javascript">
var request;

function createRequest() {
   try {
            request = new XMLHttpRequest();
            if (request.overrideMimeType) {
                     request.overrideMimeType('text/xml');
            }
   } catch (trymicrosoft) {
            try {
               request = new ActiveXObject("Msxml2.XMLHTTP");
            } catch (othermicrosoft) {
               try {
                        request = new ActiveXObject("Microsoft.XMLHTTP");
               } catch (failed) {
                        request = false;
               }
            }
   }
   if (!request)
            alert("Error initializing XMLHttpRequest!");
}
</script>
```

Communicating

- Steps
  - Gather information (possibly from HTML form)
  - Set up the URL
  - Open the connection
  - Set a callback method
  - Send the request

```
function getCustomerInfo()
{
        var phone = document.getElementById("phone").value;
        var url = "/cgi-local/lookupCustomer.php?phone=" + escape(phone);
        request.open("GET", url, true);
        request.onreadystatechange = updatePage;
        request.send(null);
}
```

Handling Server Responses

- When the server responds, your callback method will be invoked.
  - It is called at various stages of the process
  - Test readyState

```
function updatePage()
{
        if (request.readyState == 4) {
            if (request.status == 200) {
                    // Handle the response
            } else
                    alert("status is " + request.status);
        }
}
```

HTTP Ready States

- 0: The request is uninitialized
  - Before calling open()
- 1: The request is set up, but hasn't been sent
  - Before calling send()
- 2: The request is sent and is being processed
  - Sometimes you can get content headers now
- 3: The request is being processed
  - The server hasn't finished with its response
- 4: The response is complete

The XMLHttpRequest Object

- Methods
  - abort()
    - cancel current request
  - getAllResponseHeaders()
    - Returns the complete set of http headers as a string
  - getResponseHeader("headername")
    - Return the value of the specified header
  - open("method", "URL", async, "uname", "passwd")
    - Sets up the call
  - setRequestHeader("label", "value")
  - send(content)
    - Actually sends the request

The XMLHttpRequest Object

- Properties
  - onreadystatechange
    - Event handler for an event that fires at every state change
  - readyState
    - Returns the state of the object
  - responseText
    - Returns the response as a string
  - responseXML
    - Returns the response as XML - use W3C DOM methods
  - status
    - Returns the status as a number - ie. 404 for “Not Found”
  - statusText
    - Returns the status as a string - ie. “Not Found”

Typical AJAX Flow

- Make the call
  - Gather information (possibly from HTML form)
  - Set up the URL
  - Open the connection
  - Set a callback method
  - Send the request
- Handle the response (in callback method)
  - When request.readyState == 4 and request.status == 200
  - Get the response in either text or xml
    - request.responseText or request.responseXML
  - Process the response appropriately for viewing
  - Get the objects on the page that will change
    - document.getElementById or document.getElementByName, etc.
  - Make the changes

AJAX Response Handler

```
function updatePage()
{
        if (request.readyState == 4) {
            if (request.status == 200) {
                var response = request.responseText.split("|");   // “order|address”
                document.getElementById("order").value = response[0];
                document.getElementById("address").innerHTML = response[1];
            } else
                alert("status is " + request.status);
        }
}
```

jQuery
Javascript Library

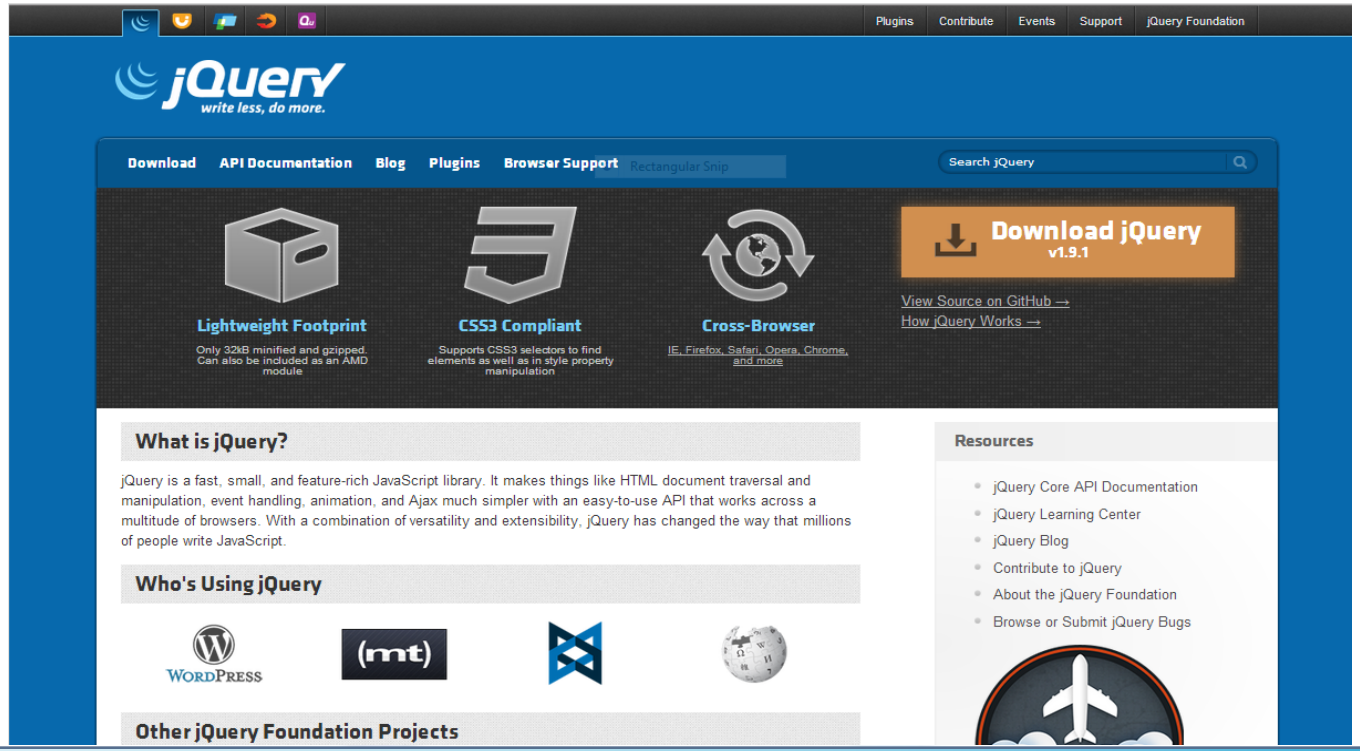
What is jQuery

- jQuery is a lightweight, open-source JavaScript library that simplifies interaction between HTML and JavaScript
- It has a great community, great documentation, tons of plugins, and also adopted by Microsoft

Download

Download the latest version from

http://jquery.com



Reference it in your markup

```
<script src="jquery.js"/>
```

jquery.js should contain a copy of the compressed production code

You can also reference it from Google

```
<script src="//ajax.googleapis.com/ajax/libs/
                                jquery/1.9.1/jquery.min.js">
</script>
```

Or by another CDN (Content Delivery Network)

The Magic $() function

```
var el = $("<div/>")
```

Create HTML elements on the fly

The Magic $() function

```
$(window).width()
```

Manipulate existing DOM elements

The Magic $() function

```
$("div").hide();
```

Selects document elements
(more in a moment…)

The Magic $() function

```
$(function(){…});
```

Fired when the document is ready for programming.

Better use the full syntax:

```
$(document).ready(function(){…});
```

jQuery's programming philosophy is:

GET >> ACT

```
$("div").hide()

$("<span/>").appendTo("body")

$("button").click()
```

Almost every function returns jQuery, which provides a fluent programming interface and chainability:

```
$("div").show()
        .addClass("main")
        .html("Hello jQuery");
```

Three Major Concepts of jQuery

The $() function

Get > Act

Chainability

All Selector

All Selector

```
$("*")          // find everything
```

Selectors return a pseudo-array of jQuery elements

Basic Selectors

By Tag:

```
$("div")
// <div>Hello jQuery</div>
```

By ID:

```
$("#usr")
// <span id="usr">John</span>
```

By Class:

```
$(".menu")
// <ul class="menu">Home</ul>
```

Yes, jQuery implements CSS Selectors!

More Precise Selectors

```
$("div.main")    // tag and class
$("table#data") // tag and id
```

Combination of Selectors

```
// find by id + by class
$("#content, .menu")
// multiple combination
$("h1, h2, h3, div.content")
```

Hierarchy Selectors

```
$("table td")          // descendants
$("tr > td")           // children
$("label + input")     // next
$("#content ~ div")    // siblings
```

Selection Index Filters

```
$("tr:first")    // first element
$("tr:last")     // last element
$("tr:lt(2)")    // index less than
$("tr:gt(2)")    // index gr. than
$("tr:eq(2)")    // index equals
```

Visibility Filters

```
$("div:visible")// if visible
$("div:hidden") // if not
```

Attribute Filters

```
$("div[id]")                // has attribute
$("div[dir='rtl']")         // equals to
$("div[id^='main']")        // starts with
$("div[id$='name']")        // ends with
$("a[href*='msdn']")        // contains
```

Forms Selectors

```
$("input:checkbox")   // checkboxes
$("input:radio")      // radio buttons
$(":button")          // buttons
$(":text")            // text inputs
```

Forms Filters

```
$("input:checked")    // checked
$("input:selected")   // selected
$("input:enabled")    // enabled
$("input:disabled")   // disabled
```

Find Dropdown Selected Item

```
<select id="cities">
    <option value="1">Tel-Aviv</option>
    <option value="2" selected="selected">Yavne</option>
    <option value="3">Raanana</option>
</select>
```

```
$("#cities option:selected").val()
$("#cities option:selected").text()
```

SELECTORS DEMO

A Selector returns a pseudo array of jQuery objects

```
$("div").length
```

Returns number of selected elements.

It is the best way to check selector.

Getting a specific DOM element

```
$("div").get(2) or $("div")[2]
```

Returns a 2nd DOM element of the selection

Getting a specific jQuery element

```
$("div").eq(2)
```

Returns a 2nd jQuery element of the selection

`each(fn)` traverses every selected element calling fn()

```
var sum = 0;
$("div.number").each(
        function(){
          sum += ($(this).innerHTML);
        });
```

this – is a current DOM element

each(fn) also passes an indexer

```
$("table tr").each(
   function(i){
        if (i % 2)
             $(this).addClass("odd");
   });
```

$(this) – convert DOM to jQuery

i - index of the current element

Traversing HTML

```
.next(expr)      // next sibling
.prev(expr)      // previous sibling
.siblings(expr)  // siblings
.children(expr) // children
.parent(expr)    // parent
```

Check for expression

```
$("table td").each(function() {
   if ($(this).is(":first-child")) {
       $(this).addClass("firstCol");
   }
});
```

Find in selected

```
// select paragraph and then find
// elements with class 'header' inside
$("p").find(".header").show();

// equivalent to:
$(".header", $("p")).show();
```

Advanced Chaining

```
$("<li><span></span></li>") // li
    .find("span") // span
        .html("About Us") // span
        .andSelf() // span, li
            .addClass("menu") // span,li
        .end() // span
    .end() // li
    .appendTo("ul.main-menu");
```

Get Part of Selected Result

```
$("div")
        .slice(2, 5)
        .not(".green")
            .addClass("middle");
```

HTML Manipulation

Getting and Setting Inner Content

```
$(“p”).html(“<div>Hello $!</div>”);


// escape the content of div.b
$(“div.a”).text($(“div.b”).html());
```

Getting and Setting Values

```
// get the value of the checked checkbox
 $("input:checkbox:checked").val();
```

```
// set the value of the textbox
 $(":text[name='txt']").val("Hello");
```

```
// select or check lists or checkboxes
 $("#lst").val(["NY","IL","NS"]);
```

Handling CSS Classes

```
// add and remove class
$("p").removeClass("blue").addClass("red");
```

```
// add if absent, remove otherwise
$("div").toggleClass("main");
```

```
// test for class existance
 if ($("div").hasClass("main")) { //… }
```

Inserting new Elements

```
// select > append to the end
$("h1").append("<li>Hello $!</li>");
// select > append to the beginning
$("ul").prepend("<li>Hello $!</li>");
```

```
// create > append/prepend to selector
$("<li/>").html("9").appendTo("ul");
$("<li/>").html("1").prependTo("ul");
```

Replacing Elements

```
// select > replace
$("h1").replaceWith("<div>Hello</div>");
```

```
// create > replace selection
$("<div>Hello</div>").replaceAll("h1");
```

Replacing Elements while keeping the content

```
$("h3").each(function(){
    $(this).replaceWith("<div>"
                        + $(this).html()
                        + "</div>");
    });
```

Deleting Elements

```
// remove all children
$("#mainContent").empty();
```

```
// remove selection
$("span.names").remove();
```

```
// change position
$("p").remove(":not(.red)")
                .appendTo("#main");
```

Handling attributes

```
$("a").attr("href","home.htm");
 // <a href="home.htm">…</a>
```

```
// set the same as the first one
 $("button:gt(0)").attr("disabled",
    $("button:eq(0)").attr("disabled));
```

```
// remove attribute - enable
 $("button").removeAttr("disabled")
```

Setting multiple attributes

```
$("img").attr({
        "src" : "/images/smile.jpg",
        "alt" : "Smile",
        "width"  : 10,
        "height" : 10
 });
```

CSS Manipulations

```
// get style
$("div").css("background-color");
```

```
// set style
$("div").css("float", "left");
```

```
// set multiple style properties
$("div").css({"color":"blue",
              "padding": "1em"
              "margin-right": "0",
              marginLeft: "10px"});
```

Events

When the DOM is ready…

```
$(document).ready(function(){
   //…
});
```

- Fires when the document is ready for programming.
- Uses advanced listeners for detecting.
- window.onload() is a fallback.

Attach Event

```
// execute always
$("div").bind("click", fn);
// execute only once
$("div").one("click", fn);
```

Possible event values:

blur, focus, load, resize, scroll, unload, beforeunload, click, dblclick, mousedown, mouseup, mousemove, mouseover, mouseout, mouseenter, mouseleave, change, select, submit, keydown, keypress, keyup, error

(or any custom event)

jQuery.Event object

CONTENTS

Detaching Events

```
$("div").unbind("click", fn);
```

(Unique ID added to every attached function)

Events Triggering

```
$("div").trigger("click");
```

Triggers browser's event action as well.

Can trigger custom events.

Triggered events bubble up.

Effects

Showing or Hiding Element

```
// just show
$("div").show();

// reveal slowly, slow=600ms
$("div").show("slow");

// hide fast, fast=200ms
$("div").hide("fast");

// hide or show in 100ms
$("div").toggle(100);
```

Sliding Elements

```
$("div").slideUp();
$("div").slideDown("fast");
$("div").slideToggle(1000);
```

Fading Elements

```
$("div").fadeIn("fast");
$("div").fadeOut("normal");
// fade to a custom opacity
 $("div").fadeTo ("fast", 0.5);
```

Fading === changing opacity

Detecting animation completion

```
$("div").hide("slow", function() {
        alert("The DIV is hidden");
});


$("div").show("fast", function() {
        $(this).html("Hello jQuery");
}); // this is a current DOM element
```

Every effect function has a (speed, callback) overload

Custom Animation

```
// .animate(options, duration)
$("div").animate({
                    width: "90%",
                    opacity: 0.5,
                    borderWidth: "5px"
                    }, 1000);
```

Chaining Animation

```
$("div").animate({width: "90%"},100)
        .animate({opacity: 0.5},200)
        .animate({borderWidth: "5px"});
```

By default animations are queued and than performed one by one

Controlling Animations Sync

```
$("div")
    .animate({width: "90%"},
                {queue:false, duration:1000})
    .animate({opacity : 0.5});
```

The first animation will be performed immediately without queuing

AJAX with jQuery

Loading content

```
$.ajax({url: ”test.php",
 success: function(result) {
                $("#div1").html(result);
        }
});
```

Sending GET/POST requests

```
$.get("test.php", {id:1},
        function(data){alert(data);});


$.post("test.php", {id:1},
        function(data){alert(data);});
```

Retrieving JSON Data

```
$.getJSON("users.php", {id:1},
        function(users)
        {
            alert(users[0].name);
        });
```

Tài Liệu Tham Khảo

- [1] Stepp,Miller,Kirst. Web Programming Step by Step.(1st Edition, 2009) Companion Website: http://www.webstepbook.com/
- [2] W3Schools, http://www.w3schools.com/html/default.asp