JavaScript: DOM / jQuery

Wei-Shao Tang

May 5, 2015

Topics Covered

- How Browser Works.
- DOM and DOM API.
- jQuery Library.
  - DOM Traversal and Manipulation
  - Event-Driven Programming
  - AJAX
  - jQuery plugins (Bootstrap)
- NOTICE: This tutorial includes interactive demo, it's better to see html version if you currently aren't: `http://pa4373.github.io/jstutorials/dom_jquery.html`

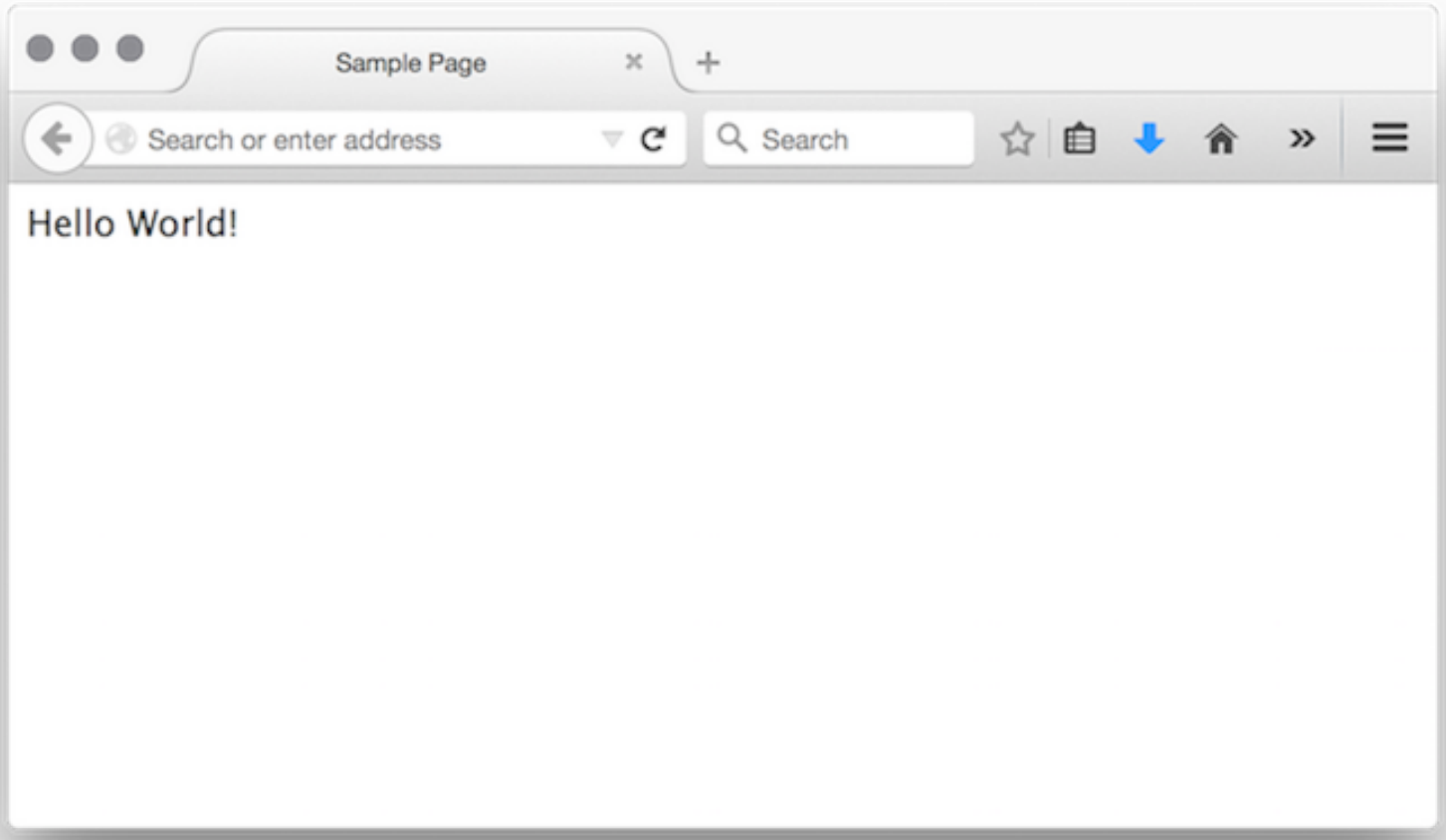
How Browser Works?

- Given the following html snippet:

```
<html lang='en'>
<head>
    <meta charset='utf-8'>
    <title>Sample Page</title>
</head>
<body>
    <div id='content'></div>
    <script type='text/javascript'>
    (function () {
      var domObj = document.getElementById('content');
      domObj.innerHTML = 'Hello World!';
    })();
    </script>
</body>
</html>
```

How Browser Works? (Cont'd)

- It can be transformed to what we see:



How Browser Works? (Partially)

- Source $\rightarrow$ (DOM Parser) $\rightarrow$ DOM Tree $\rightarrow$ (Render Engine) $\rightarrow$ What we saw.
  1. Source code (plain text) will be parsed to DOM tree.
  2. With DOM tree, browser will render content on the screen accordingly. (with CSS, if exists.)
- DOM stands for *Document Object Model*.
- **DOM Tree** is essentially **data represented in tree structure**.

DOM and DOM API

- Such design enables the possibility to use a script language to provide interaction with users.
- After DOM objects were created, the script has access to those objects via **DOM API**.
- In the context, the script is written in JavaScript.
  - There might be others such as VBScript (dead already), Java Applet...
- Remark: The program responsible for executing JavaScript codes called 'JavaScript Engine.'

DOM API (in JavaScript)

- DOM API looks like this:

```
1  var domObj = document.getElementById('content');
2  // get dom object which 'id' is 'content'
3  var newDomObj = document.createTextNode("Water");
4  // create new dom object.
5  domObj.appendChild(newDomObj);
6  // insert new node to the selected as its child.
7  domObj.addEventListener('click', function () {
8    window.alert('WHOOOPAH!');
9  });
10 // Upon clicking the text, show the alert box.
```

DOM API (in Javascript, Cont'd)

- It includes:
  - Traversal: Select DOM objects satisfying criteria.
  - Manipulation: Change its attributes, add / remove it's parent / children node and such.
  - Event Handling: **Binding** functions when certain events occur, such as *click*.
- Basically, you operate on a tree structure.
- More

How to write JavaScript in Browser?

- Either by putting codes inline in <script> tag:

```
...
    <script type='text/javascript'>
    (function () {
      your code goes here.....
    })();
    </script>
</body>
</html>
```

- or link to external file:

```
<script type='text/javascript' src='main.js'></script>
```

where JavaScript codes lie in *main.js*.

How to write JavaScript in Browser? (Cont'd)

- Where to put the script tags?
  - Well, it varies, the main concerns are:
    1. Validity of HTML
    2. Synchronous Loading (Blocking)
- Validity of HTML means your code following the standard of HTML structure.
- Synchronous Loading: When browser loads your HTML, it parses line by line, constructs DOM tree and renders accordingly.
  - If it encounters `script` tags, either it executes immediately,
  - or it loads external files.
  - All operations blocks the web page rendering.

How to write JavaScript in Browser? (Cont'd)

- Few years ago, it's wildly recommended to put script tags **at the bottom of `body` section**, since JS codes don't load until DOM tree is constructed:
  - However, if your codebase is quite large (think of Facebook), your users have to wait longer for your application initialised.
  - For small size program, this works just fine.
- For modern browsers, you can put `script` tags **at the bottom of `head` with `async` or `defer` attributes**:

```
1 <script type='text/javascript' src='app1.js' defer></script
2 <script type='text/javascript' src='app2.js' defer></script
```

- Scripts with `async` tag executes immediately upon file is loaded, which is non-deterministic, while `defer` tag ensures your files executes in sequence asynchronously.

Debug trick

- WARNING: The trick code demonstrated below shall be in development only, remove it when it's ready to go production.

1. `console.log`: Print text under the JavaScript REPL. If your input is object / array, it allows you to inspect its attribute.

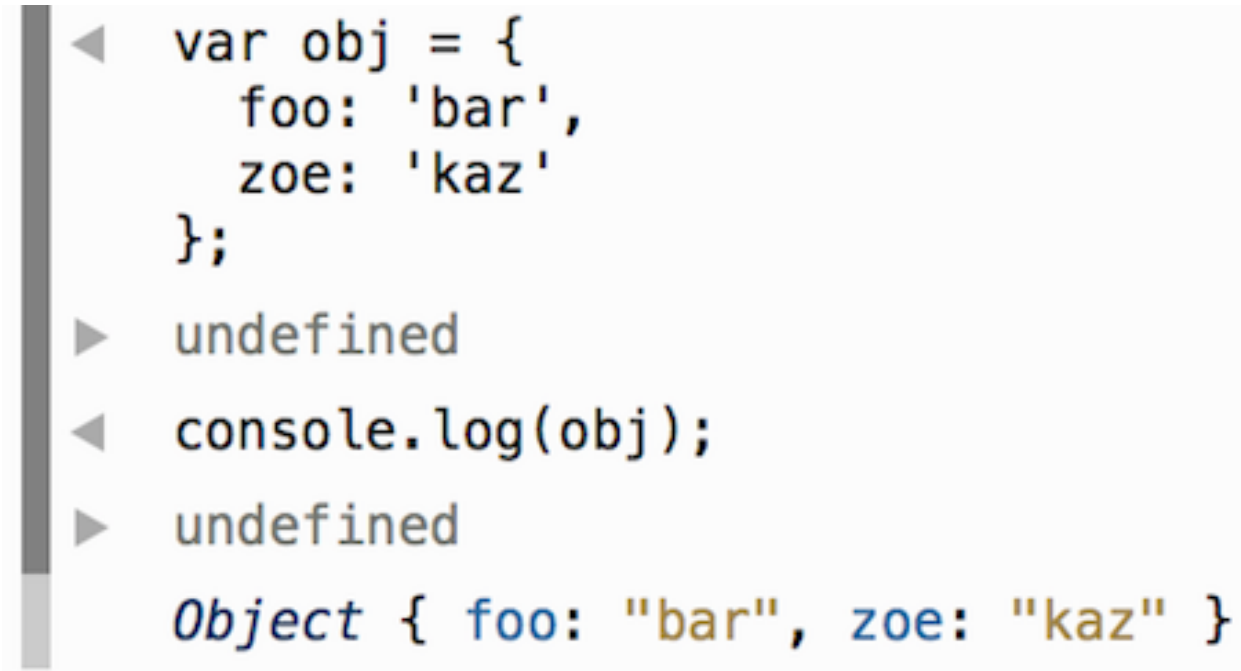


Figure 2: console.log on REPL

Debugger

- 'The formal way' to debug your application. (see the joke)
- Basic steps:
  1. Navigate to the code section you would like to inspect.
  2. Set proper breakpoints
  3. Execute program, and the program stops at breakpoints.
  4. Inspect the program state at the moment.
- Details varies in different browsers, you might want to check out yourself:
  - Firefox: `https://developer.mozilla.org/en-UK/docs/Tools/Debugger`
  - Chrome: `https://developer.chrome.com/devtools/docs/javascript-debugging`

jQuery

- DOM Traversal / Manipulation can be tedious, using native API.
- You need to pay attention to Cross-Browser compatibility.
- Writing DOM animation can be hard, at least for me. (though with the advent of CSS3 animation it changes.)

jQuery (Cont'd)

- To address those problems, jQuery was born to simplify those task:

```
1 var domObj = document.getElementById('content');
2 domObj.innerHTML = 'Hello World!';
3 // Using Native DOM API
4 $('#content').text('Hello World!');
5 // Using jQuery
```

- Originally written by John Resig, jQuery is the most popular JavaScript library in use today. (source)
- Massive plugins to create different effects.
- While experienced programmers tend to reduce their dependence on jQuery for better portability, modularity and performance, it increases the development speed.
- REMARK: jQuery is a library, and your vanilla JavaScript works just fine without it. (see the prank)

How to use jQuery

- We sticks to jQuery 2.x in this tutorial.

1. You need to include it in your page at first.
   1.1 Navigate to `https://jquery.com/download/`, download the compressed, production version.
   1.2 Let's assume the downloaded file called **jquery.min.js**, include it in your page:

```
1  <script type='text/javascript' src='jquery.min.js'></scr
```

   1.3 Put your script tag below the jQuery tag.
2. To test if it works, put the following code and test if jQuery works:

```
1  $(document).ready(function (){
2    $('body').text('jQuery works!');
3  });
```

(or just puts your download link in `src` directly.)

Basic Model

- jQuery is a giant function (object) with an identifier, usually with `$`. (`jQuery` itself works.)
- DOM:
  1. You **select** some elements.
  2. and you either **manipulate** them or **bind events** for them. (object method)
  3. (or using jQuery to create new DOM and insert to page later.)
- Not-related to DOM: Simplified helper function to work with. (function, ex: jQuery's ajax function: $.ajax)

$(document).ready(fn)

- A page can't be manipulated safely until the document is loaded.
  - We've seen this in *Synchronous Loading*.
- Instead of moving around your script tag, jQuery provides a method on *document* object, which *binds function* to event when *your page is loaded. (a.k.a ready)*:

```
1 $(document).ready(function () {
2   // your code won't run until the page is ready.
3 });
4 // your code runs as soon as loaded.
```

which can be simplified into:

```
1 $(function () {
2   // your code won't run until the page is ready.
3 });
```

- **It's good habit to put your code in the supplied function.**

DOM Traversal

- In jQuery, DOM traversal can be seen as three kinds:
  1. Ad-Hoc Selection: Using *CSS selector* to directly specify elements in documents.
  2. Relative Position Lookup: Given one node on DOM tree, find its parent / children nodes.
  3. Filtering current collections.

```
1 var contentNode = $('#content');
2 // Fetch the node with id `content`
3 contentNode.children().filter(':even')
4 .each(function () {
5   window.alert($(this).text());
6 });
7 /* Fetch its children, choose the even ones only,
8 and print out its value,
9 the form is tricky for now, but we'll see it later.*/
```

DOM Traversal (Cont'd)

- CSS Selector: Pretty much the same as what you learned in CSS3 part.
  - To inspect quickly: Fire up [Inspect Element], navigate to [Inspector], look for desired dom object and right click [Get Unique Selector] (Firefox)
- the results you got, known as **jQuery collection**, which you can manipulate later on.
- When you applied method related to DOM traversal / manipulation, jQuery returns **itself**, hence it supports **chaining**.

Function chaining:

- Recall string example:

```
'I\'m gonna be 500 Miles'
.replace('gonna be ', 'would walk ').toUpperCase();
//: "I'M WOULD WALK 500 MILES"
```

- You can see after `replace` function is applied, a new string is generated, and the new string invokes `toUpperCase` method and so on..
- Intuitively, The function acts as giant **function (object) with internal states change**, and this case applied to jQuery, too.
  - REMARK: String operation is actually object wrapper.
  - Hence we can see jQuery collection as jQuery function with references to selected objects.
  - Technical detail: You can enable chaining by return `this` (current object) in your object method. See more.

DOM Traversal (Cont'd)

- Consider the following HTML and JavaScript snippets, it converts all characters to upper case:

```
<blockquote id='hawking'>
The victim should have the right to end his life,
if he wants. But I think it would be a great mistake.
However bad life may seem, there is always something
you can do, and succeed at.
While there's life, there is hope. ~ Stephen Hawking
</blockquote>
```

```
var hawkingQuote = $('#hawking');
var text = hawkingQuote.text();
hawkingQuote.text(text.toUpperCase());
```

DOM Traversal (Cont'd)

- Checkout the demo:

[Hover over me to see demo.]

The victim should have the right to end his life, if he wants. But I think it would be a great mistake. However bad life may seem, there is always something you can do, and succeed at. While there's life, there is hope. ~ Stephen Hawking

DOM Traversal (Cont'd)

- What happened?

```
1  var hawkingQuote = $('#hawking');
2  var text = hawkingQuote.html();
3  hawkingQuote.html(text.toUpperCase());
```

- In line 1, we initialise our jQuery function with **nevigating its focus on 'hawking' object.**
  - It's **ad-hoc selection**, with jQuery collection containing **one object**.
- To extract its value, we call `text` function in line 2 and it returns;
- Then, we set its value with uppercased version of the string, in line 3.
  - Although we haven't covered DOM manipulation, yet. See `text` as content getter / setter, depending on whether the parameter is given.

DOM Traversal (Cont'd)

- Another example. It converts to upper case if the item is odd-indexed.

```
<ol id='songList'>
    <li>Led Zeppelin - Kashmir</li>
    <li>Billy Joel - Piano Man</li>
    <li>John Mayer - Gravity</li>
    <li>The Black Angels - Young Men Dead</li>
    <li>John Mayer - Slow Dancing In A Burning Room</li
    <li>Jimi Hendrix - Voodoo Chile</li>
</ol>
```

```
$('#songList').children().filter(':even')
.each(function (idx, ele) {
  var newText = $(ele).text().toUpperCase();
  $(ele).text(newText);
});
```

DOM Traversal (Cont'd)

- Check out the demo:

[Hover over me to see demo.]

1. Led Zeppelin - Kashmir
2. Billy Joel - Piano Man
3. John Mayer - Gravity
4. The Black Angels - Young Men Dead
5. John Mayer - Slow Dancing In A Burning Room
6. Jimi Hendrix - Voodoo Chile

DOM Traversal (Cont'd)

- What happened?

```
1  $('#songList').children().filter(':even')
2  .each(function (idx, ele) {
3    var newText = $(ele).text().toUpperCase();
4    $(ele).text(newText);
5  });
```

- In line 1, we select `songList` item, get its children nodes, and then convert some to upper case.
  1. For $('#songList'), it's standard ad-hoc selection with one object in jQuery collection. (`ol`)
  2. then we use `children` method to get its children, it's relative position lookup, resulted in multiple objects in collection. (multiple `li`)

DOM Traversal (Cont'd)

- What happened?

```
1  $('#songList').children().filter(':even')
2  .each(function (idx, ele) {
3    var newText = $(ele).text().toUpperCase();
4    $(ele).text(newText);
5  });
```

- In Line 2, we choose the even elements, using `filter`, resulted in the subset of the previous collection. (multiple `li`)
- Then we *ITERATE* the collection, using `each` method.

DOM Traversal (Cont'd)

- Why not `for (; ;)` loop?
  - Well, because what you got is not an array or any array-like object.
- However, it suffice to provide equivalent effect as `for` loop:
  - inx: the current index of the object, starting from 0.
  - ele: the current item. To use with jQuery, pass it to $ again.
- Conceptual Analogy:

```
1  $coll.each(function (inx, ele) {
2    // code block....
3  });
4
5  for (var inx = 0; inx < $coll.length; inx = inx + 1) {
6    var ele = $coll[inx];
7    // code block....
8  }
9  // Note this won't work in real case.
```

DOM Traversal (Cont'd)

- In contrast to `children` method, `parents` method gets the parent node of the current node.
- For complete ad-doc selection, consult: `http://api.jquery.com/category/selectors/hierarchy-selectors/`
- For filtering, consult: `http://api.jquery.com/category/traversing/filtering/`
- For complete tree traversal, consult: `http://api.jquery.com/category/traversing/`
- This concludes the DOM traversal chapter.

DOM Manipulation

- After you located your target collection, you can:
  1. Read / Modify its **CSS class** and **CSS style**.
  2. Read / Modify its attribute. (href, src . . .)
  3. Insert / Remove element around them.
  4. Use some built-in animations.
- Based on those operations, you can even use **plugins** to create astonishing effects with little effort.

DOM Manipulation (Cont'd)

- Given the following HTML, CSS, and JavaScript snippets:

```
<blockquote id='russell'>
A man would not say "I hope the first President of the
United States will be the first President of the United
States" unless he had an unusual passion for the law of
identity. ~ Bertrand Russell, Western Philosophy
</blockquote>
```

```
.quote-border {
  border: 1px #777 dashed;
  margin: 5px;
}
```

```
$('#russell').addClass('quote-border');
```

DOM Manipulation (Cont'd)

- See the live demo:

[Hover over me to see demo.]

A man would not say "I hope the first President of the United States will be the first President of the United States" unless he had an unusual passion for the law of identity. ~ Bertrand Russell, Western Philosophy

DOM Manipulation (Cont'd)

- It's obvious that `addClass` will append new class to elements `class` attribute.
- The opposite of `addClass` is `removeClass`.

DOM Manipulation (Cont'd)

- Given the following HTML and JavaScript snippets:

```
1  <img src='assets/hover1.jpg' id='barney'>
```

```
1  var barney = $('#barney');
2  var original = barney.attr('src');
3  // original will hold 'assets/hover1.jpg'
4  barney.attr('src', 'assets/hover2.jpg');
```

- `attr` is also a setter / getter, depending on whether the parameter is given.
- in line 4, it changes its value to `assets/hover2.jpg`

DOM Manipulation (Cont'd)

- See the demo:

[Hover over me to see demo.]

DOM Manipulation (Cont'd)

- Given the following HTML, CSS and JavaScript snippets:

```
<div id='source'>
  Come back you little bastard!
</div>
<div id='target'>
  Mom, I'm here!
</div>
```

```
#target {
  border: 1px #777 solid;
}
```

```
var source = $('#source')
$('#target').append(source);
```

DOM Manipulation (Cont'd)

- See the live demo: *[Hover over me to see demo.]*
  Come back you little bastard!
  Mom, I'm here!

- `append` method will inject elements (selected by $) inside the object.
- The opposite is `remove`.
- If you want to inject a copy instead of original one, use `$('#source').clone()` instead.
- Some other common methods are: `after`, `before`....

DOM Manipulation (Cont'd)

- **Animation**: Some jQuery methods encapsulating underlying css / time operations.
- `fadeIn`, `fadeOut`: elements fade in / away in the given time interval.
- `show`, `hide`: elements show / hide immediately.

DOM Manipulation (Cont'd)

- Consider the following HTML, Javascript snippets:

```
<div id='fade-away'>
  I'm gonna be gone.
</div>
```

```
$('#fade-away').fadeOut(1000, function () {
  window.alert('BOOM!');
});
```

DOM Manipulation (Cont'd)

- See the live demo:

[Hover over me to see demo.]

I'm gonna be gone.

- It shows that `#fade-away` object will fade out in 1000 milliseconds.
  - It can also be `slow`, `fast` string.
- After it's completely gone, the callback will trigger, showing one alert box.
- Callback function is optional, you can disable it by simply not giving a callback function.

DOM Manipulation (Cont'd)

- For all manipulation API, consult:
  `http://api.jquery.com/category/manipulation/`

Event-driven Programming

- To make your program interact with users, your program need to know how to respond user action.
- **Event**: Certain **action** occurring upon certain **elements**. (ex: `click` on `#sampleObject`)
- **Handler**: The piece codes executed when some events are triggered, usually represented in the form of function in JavaScript.
- You actually have seen that in previous example, while all demo programs are *triggered* by *hovering* your mouse over certain *area*.
- Some rules about event-driven programming:
  - Bind / unbind function to elements' event.
  - Event bubbling.
  - Avoid conflict.

Event-driven Programming (Cont'd)

- Consider the following HTML, JavaScript snippets:

```
<div id='binding-target'>
Can't click me.
</div>
```

```
$('#binding-target').html('Can click me.')
                    .on('click', function (e) {
  $(this).html('I was hit! not again plz!');
  window.alert('Event Type: '+ e.type);
  return false;
});
```

Event-driven Programming (Cont'd)

- See the live demo:

[Click me to bind]

Can't click me.

Event-driven Programming (Cont'd)

- `.on( events [, selector ] [, data ], handler )`
- `on` method registers function to events. (such as `click`, `hover...`)
  - Function invoked when certain event is triggered.
  - Apart from browser built-in events, you can design your own event.
- In your handler function, you can supply `e` as event object.
- The opposite of `on` is `off`
- Minor: There's another way to invoke which supplies children selector. It works by delegating your event to those matched selection.
- API: `http://api.jquery.com/on/`

Event-driven Programming (Cont'd)

- DOM elements usually nested inside each other, the handler of the parent works even if you click on it's child.
- Event bubbling: After an event triggers on the deepest possible element, it then triggers on parents in nesting order.
- Sometimes it may cause problem,
  - For example, in this slide (web version), you go to the next by clicking, imagine what will happened if I didn't take precaution (in which case of course I did, self five!) in the previous example.

Event-driven Programming (Cont'd)

- Remember `e` (as event object) in handler function?
- It provides two methods:
  - `stopPropagation`: It stops itself from bubbling to the parent.
  - `preventDefault`: It prevent its default action from happening. (such as form submitting.)
- `return false` = `stopPropagation` + `preventDefault` + `return`. (returning value does no good in handler function, right?)

Event-driven Programming (Cont'd)

- In time, you may develop more complicated program require dynamically binding / unbinding.
- It's good habit to unbind old handler before new one is attached:

```
1 $('#target-binding').off('click')
2                     .on('click', function () {
3 ...});
```

- To avoid conflict and strange behaviour, best keep your logic simple. (think about, is there cleaner way to express equivalent program logic?)

Event-driven Programming (Cont'd)

- Shortcut: `on('click', handler)` can be expressed as `click(handler)`
- This applies to : `change` (when value of form changes), `submit` (right before the form is submitted), and such..

Event-driven Programming (Cont'd)

- Actually, `ready` is also a binding, (`$(document).ready`)
- For all available event, consult API:
  `https://api.jquery.com/category/events/`

HTTP Protocol

- Application protocol underlying browser communication.
- Request: client constructs a request to server. (along with information.)
- Response: server will send a response to client.

HTTP Protocol

```
*   Trying 193.203.65.45...
* Connected to pinewoodpictures.com (193.203.65.45) port 80
> GET /film/spooks-the-greater-good/ HTTP/1.1
> Host: pinewoodpictures.com
> User-Agent: curl/7.42.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 04 May 2015 16:14:18 GMT
< Server: Apache
< X-Powered-By: PHP/5.4.35
< X-Pingback: http://pinewoodpictures.com/xmlrpc.php
< Link: <http://pinewoodpictures.com/?p=292>; rel=shortlink
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
<
<!doctype html> .... content omitted.
```

HTTP Protocol

- The first part is your request header, and the second is the response.
- Request Method: what kind of operation you intended to do on server ('GET', 'POST', and etc. . .)
- Status Code: The response code indicates the status of the operation (200 for OK, 404 for not found. . .)
- You can inspect the information within the browser, using [Inspect Element] > [Network]

AJAX

- AJAX stands for Asynchronous JavaScript and XML.
- Instead of one specific technology, AJAX is composed of several different technologies, and the use of XML is not a must.
- Intuitively, it use JavaScript code to retrieve partial content from server (string mostly), and do something with the retrieved content.
  - Hence it frees us from constant reloading and better desktop-alike user experience.

AJAX

- Generally, the AJAX procedure in JavaScript goes like:
  1. Prepare your HTTP request: specify the request url, method (`GET`, `POST`..), request parameter and callback function.
  2. After you send your request, you let your callback function to handle the rest. (known as asynchronous programming)
  3. If it got data from server, usually in the form of JSON (or XML), the callback function is invoked.
     - JSON (or XML) is plain string able to be converted (deserialise, with `JSON.parse`) to data structure, or the other way around (serialise, with `JSON.stringify`).
     - JSON, JavaScript Simple Object Notation: boolean, number, string, array, object and friends. (sounds familiar?)
     - You can operate on the data structure in the callback function after that.

Serialisation / Deserialisation of JSON

```
1  var obj = {
2    foo: 'bar',
3    zoo: 'car'
4  };
5  var jsonString = JSON.stringify(obj);
6  jsonString;
7  //: "{"foo":"bar","zoo":"car"}"
8  typeof jsonString;
9  //: "string"
10 var obj2 = JSON.parse(jsonString);
11 obj2;
12 //: Object { foo: "bar", zoo: "car" }
```

$.ajax

- To do AJAX in native browser API can be really tedious. (browser compatibility mostly.)
- jQuery provides `\$.ajax` function to simplify the task.
- API: `http://api.jquery.com/jquery.ajax/`

$.ajax

- `jQuery.ajax( url [, settings ] )`
  - Note there's no selection this time.
- `url`: the request url.
- `settings` is a object containing the detail of the request and callback. Some common ones are:
  - `method`: HTTP method you would like to perform, such as 'GET', 'POST'
  - `data`: the parameters sent along with the request, can be object ({a: 0, b: 1}), parameter string ('a=0&b=1')
  - `success`: A function to be called if the request succeeds.
    - `Function( Anything data, String textStatus, jqXHR jqXHR )`
    - success if HTTP status code in the range of 200 to 299, or 304.
  - `error`: A function to be called if the request fails
    - `Function( Anything data, String textStatus, jqXHR jqXHR )`

$.ajax

- - `complete`: A function to call after `success` and `error` is called.
  - `headers`: You can put customised HTTP headers here. (usually helpful when it comes to authorisation..)

$.ajax

- Example: `http://api.hostip.info/get_json.php`
- Please use browser to open it, you shall see:

```
{"country_name":"TAIWAN",
 "country_code":"TW",
 "city":"Taipei",
 "ip":"140.112.102.93"}
```

- This can be seen as `data` object in the success callback function later.

$.ajax

- Considering the following HTML and JavaScript:

```
<button id='what-my-ip'>What is my IP?</button>
```

```
$('#what-my-ip').click(function () {
  $.ajax('http://api.hostip.info/get_json.php', {
    method: 'GET',
    success: function (data) {
      window.alert('Hello sir, you came from ' +
      data.city + ', ' + data.country_name +
      ', and your IP address is ' + data.ip);
    }
  });
  return false;
});
```

$.ajax

- See the live demo:

What is my IP?

$.ajax

- $.ajax is the core low-level interface in jQuery, there are some shortcuts available:
  - $.get, $.getJSON, $.post...
- Consult API: `http://api.jquery.com/category/ajax/shorthand-methods/`
- Notice in Chrome, you need to put your website on the server or AJAX won't work.
- To debug with remote API, some tools comes handy:
  - Postman - REST Client: a tool on Google Chrome to inspect RESTful API.
  - RESTClient, a debugger for RESTful web services.: Firefox people's Postman.
  - curl: hardcore unix programmer use it. (not me.)

jQuery Plugins (Bootstrap)

- jQuery plugins extends jQuery's functionality to allow you do much with little effort.
- It's impossible to cover all kind use of plugins, so the really important thing is to know how to read plugins manual instead.
- Some recommended tips:
  1. Read `Get started`, `Quick start` to get the plugin run first.
  2. If you can't directly dump in API doc, see if some tutorial available.
  3. Read API doc carefully to know how to use.
- For some non-trivial plugin, it usually involves some mental model to understand. Get it first and you will learn how to use it more efficiently.
- Here, we takes `Bootstrap` jQuery plugins for example: `http://getbootstrap.com/javascript/`