AJAX

WS Slides #4

Content

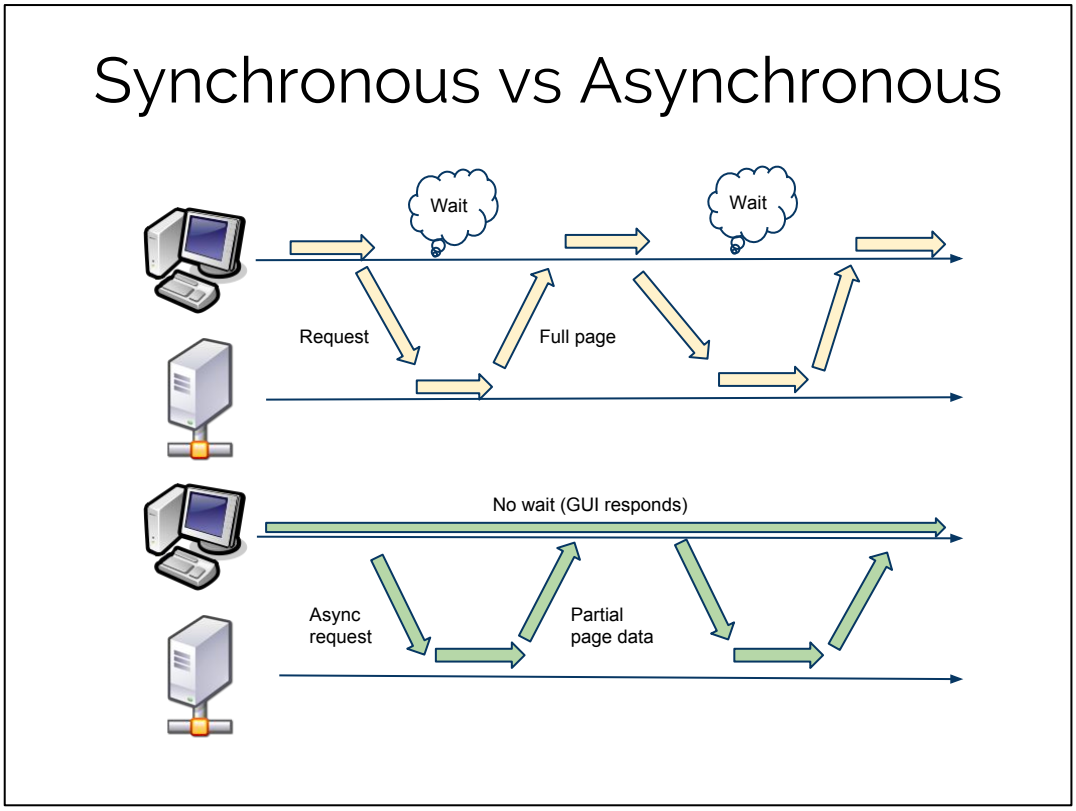
- JSON
- AJAX
- XMLHttpRequest
- Asynchronous programming
- Promises
- jQuery and Angular AJAX
- Single page application
- Bookmarkability
- Same origin Restriction

JSON

```
{
    "firstName": "John",                    // Note " not '
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"     // integer 10021 also
possible!
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }]}
```

The JavaScript Object Notation (JSON) data interchange format

- JavaScript Object Notation (JSON) is a text format for the serialization of structured data.
    - It is derived from the object literals of JavaScript
- JSON can represent four primitive types (strings, numbers, booleans and null) and two structured types (objects and arrays).
- An Object is an unordered collection of zero or more name/value pairs
    - A name is a string and a value is a string, a number, a boolean, null, an object, or an array
    - An array is an ordered sequence of zero or more values
- Normally automatic conversion from/to JSON /JavaScript object in browser at request and response
- If not converted, use global JSON-object (JSON.parse/JSON.stringify)

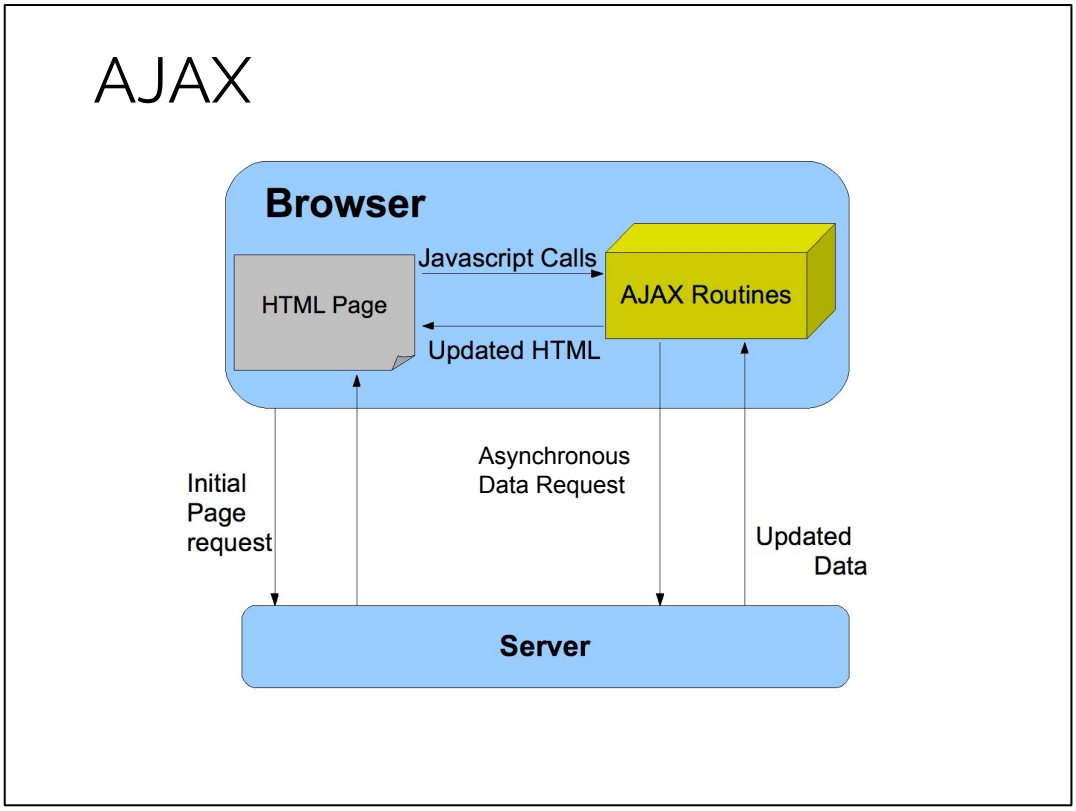


Synchronous request

- User must wait while new page or content loads

Asynchronous request

- The "asynchronous" word, means that the response from the server will be processed when available, at some later time
- Request will return directly (main program will continue)
    - Client will not wait and freeze the display of the page.



AJAX (or AJAX, short for asynchronous JavaScript and XML)

- Is a group of interrelated Web development techniques used on the client-side to create asynchronous Web applications.
- With Ajax, web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page

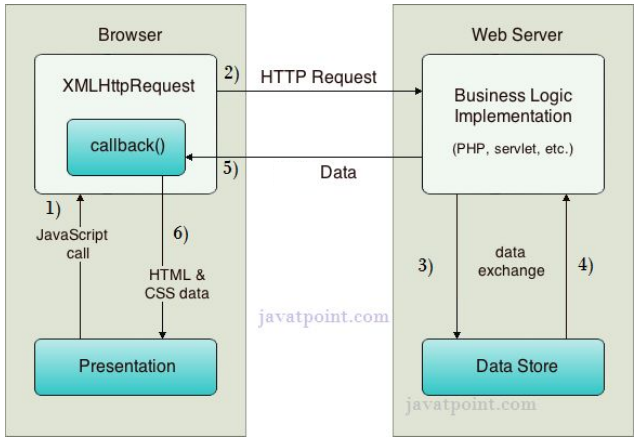
Initial request - Full page load

- A full DOM is constructed
- Browser address bar will change

Asynchronous request - updated page (partial page load)

- New data (not full page) delivered to client
  - Inserted into existing DOM
- No change in address bar

XMLHttpRequest



The XMLHttpRequest object is an API for fetching resources (and also POST) .

- The object supports any text based format, including XML, JSON.
- It can be used to make requests over both HTTP and HTTPS
- API is low level, we avoid

JS single threaded, how to make calls "in background"?

- Solution is to provide a callback function to the XMLHttpRequest objects asynchronous method (as a parameters)
  - Asynchronous calls has no return values.
- Callback automatically executed when async request data ready
- This is know as : Asynchronous programming

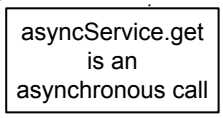
Calls to XMLHttpRequest will progress through "ready states" (setting readyState of XMLHttpRequest object)

- Open -> readyState = 1
- After send -> readyState = 2
- Response content begins to load -> readyState = 3
- Finished loading -> readyState = 4
- Each change will trigger call to handler function assigned to onreadystatechange attribute (possible to display progress)

NOTE: Chrome Developer Tools > Network has tab for XHR (XMLHttpRequest)

Asynchronous Programming

```
$(function () {
    $("#get").on("click", function () {
        asyncControl.get(function () {
            asyncService.get((function () {
              $("#result").html(JSON.
                 stringify(this.
                      req.responseText));
            }).bind(asyncService));
        });
    });
});
```



Asynchronous programming may lead to "callback hell"

- Callback hell is the eventual state of any complex callback oriented asynchronous program where your code becomes extremely hard to understand, reason about and maintain.
- Too much of nested anonymous inline callbacks

Promises

Callback approach

```
async1(function(){
    async2(function(){
        async3(function(){
            ....
        });
    });
});
```

Promise approach

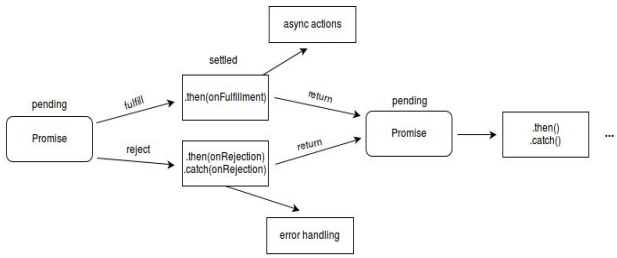
```
var task1 = async1();
var task2 = task1.then(async2);
var task3 = task2.then(async3);

task3.catch(function(){
    // Errors from task1, task2, task3 here
})
```

The Promise object (new in ECMA 6 but available already in ECMA 5) is used for deferred and asynchronous computations.

- A Promise represents an operation that hasn't completed yet, but is expected in the future.
    - It allows you to associate handlers to an asynchronous action's eventual success value or failure reason.
    - This lets asynchronous methods return values like synchronous methods: instead of the final value, the asynchronous method returns a *promise* of having a value at some point in the future.
- Promises makes async code "flat" (vs nested)
- Throw and Catch exceptions

Chaining Promises




```
var FlightDashboard = function( $scope, user, flightService, weatherService ){
        travelService
            .getDeparture( user )             // Request #1
            .then( function( departure ){
                $scope.departure = departure; // Response Handler #1
                return travelService.getFlight( departure.flightID );// Request #2
            })
            .then( function( flight ){
                $scope.flight = flight;    // Response Handler #2
                // Request #3
                return weatherService.getForecast( $scope.departure.date );
            })
            .then( function( weather ){
                $scope.weather = weather;  // Response Handler #3
            });
        };
```

Possible to chain asynchronous calls

A Promise is in one of these states

- pending: initial state, not fulfilled or rejected
- fulfilled: meaning that the operation completed successfully.
- rejected: meaning that the operation failed.

jQuery AJAX

GET (HTML)

```
$.load( ...  URL ... );
```

Get JSON

```
$.getJSON( ... URL ... );
```

Low level API

```
$.ajax({
  url: "test.html",
  context: document.body
}).done(function() {
  $( this ).addClass( "done" );
});
```

jQuery object ($) has many methods

- Call will return a Promise

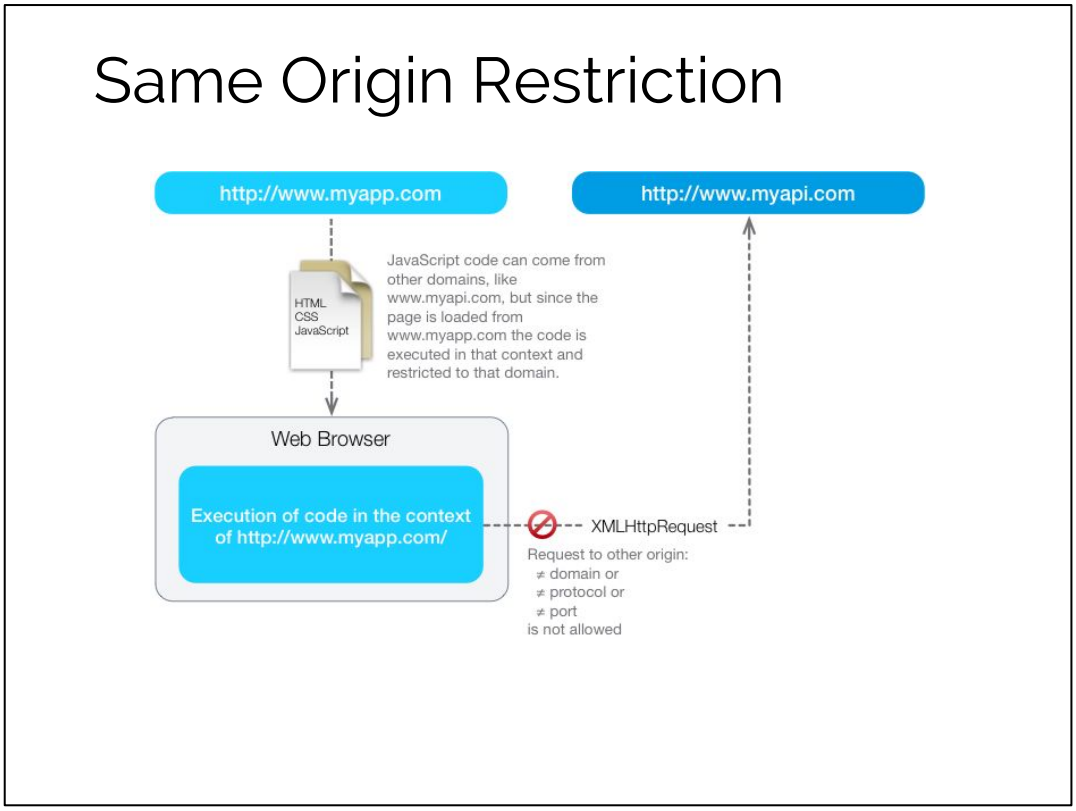
Angular AJAX

Using $http object

```
ajaxApp.factory('Proxy', ['$http',
   function ($http) {
      var url = "http://localhost:8084/ajax/...";
         return {
            get: function () {
               return $http.get(url);
            }
         };
   }]);
```

Angular has built in$http object.

- Injected when needed
- Will return Promise

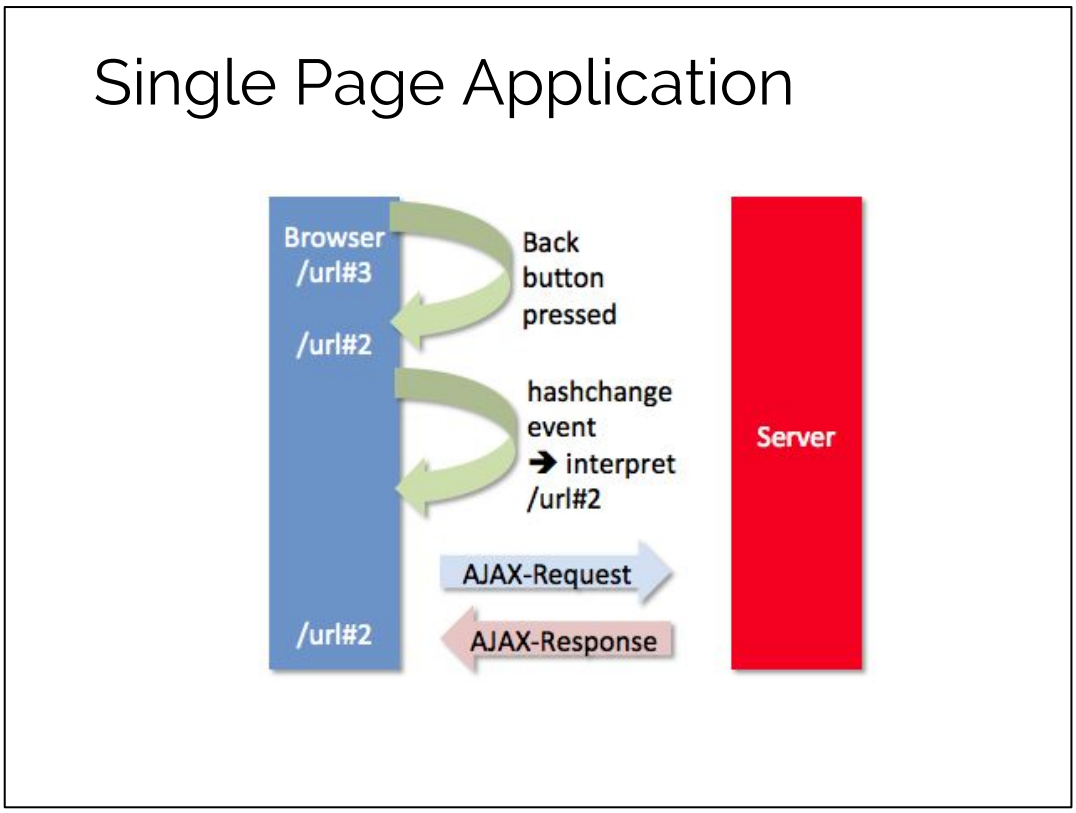


Web origin

- User agents allow content retrieve from one origin to interact freely with other content retrieved from that origin, but user agents restrict how that content can interact with content from another origin.
- Untrustworthy script in one window could use DOM methods to read the contents of documents in other browser windows, which might contain private information.

Same origin policy

- The same-origin policy restricts how a document or script loaded from one origin can interact with a resource from another origin. It is a critical security mechanism for isolating potentially malicious documents.
- The same-origin policy specifies trust by URI
- Roughly speaking, two URIs are part of the same origin (i.e., represent the same principal) if they have the same scheme, host, and port.
- A script can read only the properties of windows and documents that have the same origin as the script itself.
- NOTE: Possibly problems with local files (file:/// …)
- Poses particular problems for large websites with many servers
- JSONP, trick to circumvent restrictions
- CORS, better mechanism to allow resources from different domains
- See also X-FRAME-Options



Using AJAX we don't need any page loads, application stays on same page

Problems with AJAX (single page applications)

- If using AJAX the URI want change
  - Normally uses # to navigate (but not sent to server)
    - HTML5 hashChange-event
  - Problems with bookmarks, forward/backward, favorites
- Complex to fix, better use 3rd party libraries