

JavaScript, jQuery and Ajax

Lab. Bases de Dados e Aplicações Web

MIEIC, FEUP 2017/18

Sérgio Nunes
DEI, FEUP, U.Porto

The Big Picture

JavaScript is the programming language
of the web browser

Client-Side Technology

- HTML > content & structure.
- CSS > presentation & layout.
- JavaScript > dynamics & behavior.

Context

- Client-side scripting language that runs on web browsers.
Interpreted and object-oriented language.
- Joint effort by Netscape and Sun (circa 1995).
- Microsoft developed JScript.
Macromedia developed ActionScript.
- ECMAScript is the standard specification
developed by Ecma International.
- Unrelated to Java.

Use Cases

- Modify appearance of a web page.
- Alter the content of a web document.
- Respond to user interactions.
- Retrieve and present remote information.
- ...

JavaScript Code

```
function do() {  
    var foo = "Hello World!";  
    var bar1 = 1;  
    var bar2 = 2;  
  
    var text = foo + bar1 + bar2;  
  
    console.log(text);  
}
```

Comments in JavaScript

- JavaScript supports both single-line comments and multi-line comments.

```
function doO {  
    // This is a single line comment.  
    alert("Hello World!");  
    /* This is a  
       multi-line  
       comment.  
    */  
}
```

JavaScript Intro

Using JavaScript

- JavaScript can be included in a web document using three approaches:
 - Included from an external file (URL).
 - Included inline in the HTML code.
 - Included as a HTML element attribute.

Embedded JavaScript

- JavaScript can be attached to HTML elements using event handlers such as onclick, onmousedown, onmouseup, etc.
- No separation of concerns. Lower readability.
- Code cannot be reused. Leads to repetition.
- Bad practice that should be avoided.

```
<a href="#" onclick="alert('Clicked!')">
```

Inline JavaScript

- JavaScript code can be embedded within the HTML document using the script element.
- JavaScript code is executed as page loads.

```
<h1>Page Title</h1>
<script type="text/javascript">
    document.write("Print this text!");
</script>
<p>Yada, yada, yada!</p>
```

External JavaScript

- JavaScript may be defined in an independent file and then attached to multiple HTML documents.
- Promotes code reusability and modularity.
- Results in cleaner and readable code.
- Best option in most scenarios.

```
<head>
  ...
  <script type="text/javascript" src="file.js"></script>
  ...
</head>
```

Primitive Types

- Numbers: 23, 2.3, .23.
- Strings: "a string", 'another string'.
- Booleans: true, false.
- Undefined: undefined.
- Null: null.
- Numbers, strings and booleans are object-like in that they have methods.

Variables

- JavaScript is a dynamically typed language. No need to specify the data type in variable declaration. Data types are converted automatically as needed during execution.
- Variables must start with a letter, an underscore or a dollar sign (\$). Variable names are case sensitive.
- Variables are declared with the var keyword.

```
var userName;  
var eMail = "foo@bar.com";  
var age = 25;  
  
userName = "John Doe";  
age = "fish!";
```

Expressions

- An expression is any valid set of literals, variables, operators, and expressions that evaluate to a single value.
- JavaScript has four types of expressions:
 - Arithmetic: evaluates to a number.
 - String: evaluates to a string.
 - Logical: evaluates to true or false.
 - Object: evaluates to an object.

Strings

- Strings can be joined together using the concatenation operator (+).
- The shorthand assignment operator (+=) also works with strings.
- Strings have properties and methods.

```
>"hello".length
```

```
5
```

```
>"hello".charAt(0)
```

```
h
```

```
>"hello world".replace("hello", "goodbye")
```

```
goodbye world
```

```
>"hello".toUpperCase()
```

```
HELLO
```

Assignment Operators

- An assignment operator assigns a value to its left operand based on the value of its right operator.

Shorthand Operator	Meaning
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x %= y$	$x = x \% y$
$x ^= y$	$x = x ^ y$

Comparison Operators

- A comparison operator compares its operand and returns a logical value based on whether the comparison is true.

Operator	Description
Equal (==)	Returns true if the operands are equal.
Not equal (!=)	Returns true if the operands are not equal.
Strict equal (===)	Returns true if the operands are equal and of the same type.
Strict not equal (!==)	Returns true if the operands are not equal and/or not of the same type.
Greater than (>)	Returns true if the left operand is greater than the right operand.
Lower than (<)	Returns true if the left operand is lower than the right operand.

Arithmetic Operators

- Arithmetic operators take numerical values as their operands and return a single numerical value.
- Standard arithmetic operators work as in other languages: addition (+), subtraction (-), multiplication (*), and division (/).

Operator	Description
Modulus (%)	Returns the integer remainder of dividing the two operands.
Increment (++)	Adds one to its operand.
Decrement (--)	Subtracts one from its operand.
Unary negation (-)	Returns the negation of its operand.

Logical Operators

- Logical operators are typically used with Boolean values.

Operator	Description
&&	Logical AND. Returns true if both operands are true, otherwise false.
	Logical OR. Returns true if either operand is true. If both are false, returns false.
!	Logical NOT. Returns false if operand can be converted to true, otherwise false.

Objects

- JavaScript objects are simply collections of name-value pairs.

```
var obj1 = new Object();
// or
var obj2 = {};
```

```
obj.name = "John";
obj.surname = "Doe";
obj.age = 40;
// or
obj["name"] = "John";
obj["surname"] = "Doe";
obj["age"] = 40;
```

Arrays

- In JavaScript arrays are a special type of objects.
- Arrays support several methods:
 - a.pop() — removes and returns the last item.
 - a.push(item, ...) — adds one or more items to the end.
 - a.reverse() — returns a new array with items in reverse order.
 - a.slice(start, end) — returns a sub-array.
 - etc.

```
var cars = new Array();
cars[0] = "Ford";
cars[1] = "Opel";
cars[2] = "BMW";
```

```
var cars = ["Ford", "Opel", "BMW"];
for (var i = 0; i < cars.length; i++) {
  document.write( cars[i] + " ");
}
```

Statements

Block Statements

- Block statements are used to group statements together. The block is delimited using curly brackets.

```
{  
    statement_1;  
    statement_2;  
    ...  
    statement_n;  
}
```

Conditional Statements

- A conditional statement is a set of commands that executes if a specified condition is true.

if..else statement

```
if (condition) {  
    statements  
} [else {  
    statements  
}]
```

switch statement

```
switch (expression) {  
    case label1:  
        statements  
        break;  
    case label2:  
        statements  
        break;  
    default:  
        statements  
}
```

Loop Statements

- Loop statements executes repeatedly until a specified condition is met.
- JavaScript supports several types of loops: for, do ... while and while.

for

```
for (var i = 0; i < 5; i++) {  
    // Will execute 5 times.  
}
```

while

```
var n = 0;  
while (n < 3) {  
    n++;  
    document.write(n);  
}
```

do ... while

```
var i = 0;  
do {  
    i += 1;  
    document.write(i);  
} while (i < 5);
```

label, break and continue

- A label can be used to associate a statement with an identifier, which can be referenced elsewhere in the program.
- The break command is used to terminate a loop, switch or label statement.
- The continue statement can be used to restart a for, do...while, while, or label statement. It can be used with or without a label.

Object Manipulation Statements

- JavaScript supports for...in, for each...in, and with to manipulate objects.
- The for...in statement iterates a specified variable over all properties of an object. For each distinct property, executes the specified code. The for each...in statement is similar to for..in but iterates over the values of the object's properties, not their names.
- The with statement establishes the default object for a set of statements.

```
var cars = ["Ford", "Opel", "BMW"];  
  
for (x in cars) {  
    document.write(cars[x] + " ");  
    // Ford Opel BMW  
}
```

```
var a = 0;  
  
with (Math) {  
    a = PI + cos(PI) * sin(PI);  
}
```

Functions

- Functions are a core element in JavaScript.
- Functions are defined using the function keyword followed by the name of the function, the list of arguments separated by commas and enclosed in parentheses, and the function statements enclosed in curly braces.
- It is possible to create anonymous functions.
This makes possible to use functions as standard expressions.

```
function square (number) {  
    return number * number;  
}
```

Standard function

```
alert((function (number) {  
    return number * number;  
} )(5));
```

Anonymous function

Exception Handling

- With JavaScript it is possible to throw exceptions with the throw command and handle them using the try...catch command. A finally block statement can exist to be executed after the try...catch block.

```
throw "Error2";
throw 42;
throw true;
```

```
openXYZ();
try {
    // try something
} catch (e) {
    // handle problems
} finally {
    closeXYZ();
}
```

```
try {
    // function could throw two exceptions
    functionXYZ(); }

catch (e if e == "InvalidNameException") {
    // call handler for invalid names }

catch (e if e == "InvalidIdException") {
    // call handler for invalid ids }

catch (e) {
    // or else...
}
```

jQuery Overview

jQuery

- jQuery is a cross-browser JavaScript library developed by John Resig (2006).
- jQuery philosophy is "Write less, do more".
- Main goal: simplify client-side scripting.
- Free and open-source.
- Powerful, small and easy to use.

Reasons for jQuery Success?

- Leverage knowledge of CSS.
- Support extensions.
- Abstract away browsers quirks.
- Slim (less than 20KB compressed).
- Free and open-source.

What jQuery does?

- Access elements in a document.
- Modify the appearance of a web page.
- Alter the content of a document.
- Respond to a user's interactions.
- Animate changes being made to a document.
- Retrieve information from a server with page refresh (aka AJAX).
- Simplify common JavaScript tasks.

jQuery Setup

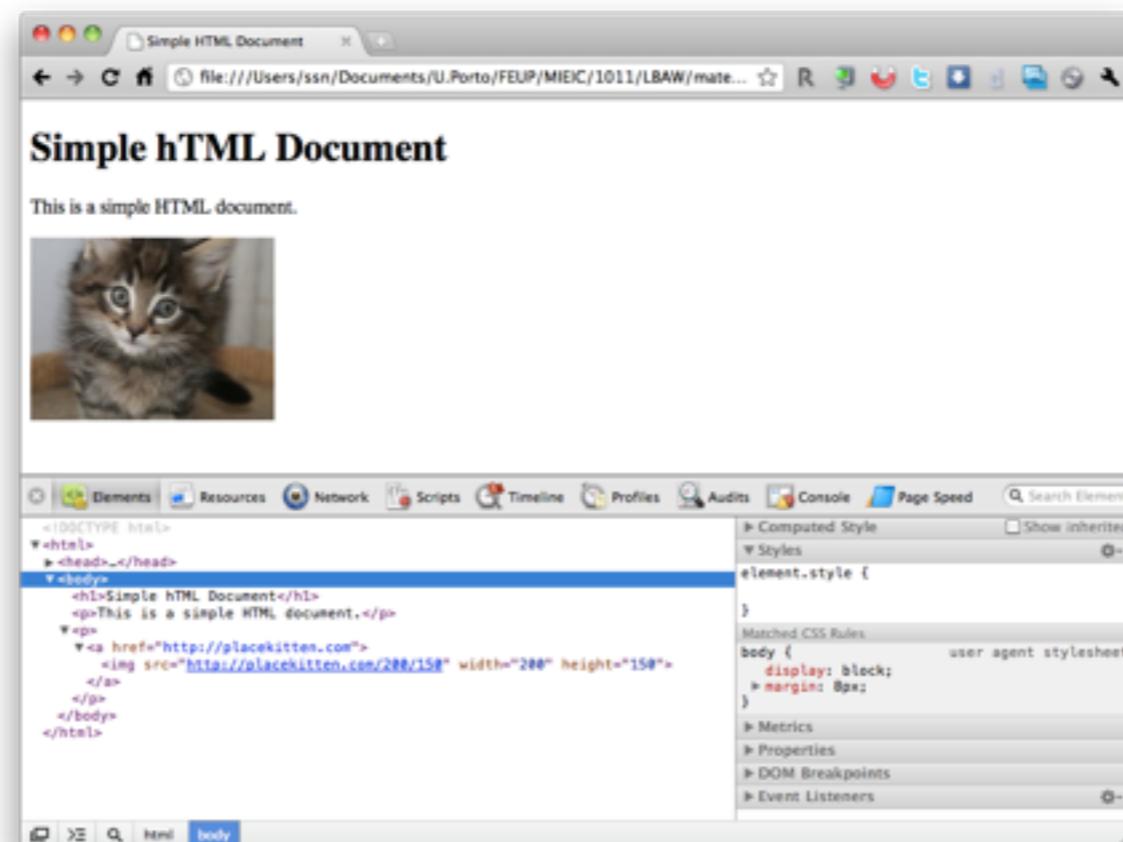
- The jQuery library needs to be loaded to be used in a HTML document.
- It can either be downloaded and served locally or accessed remotely.
- Download from <http://jquery.com>.

```
<script type="text/javascript" src="/javascript/jquery.js"></script>
```

```
<script type="text/javascript" src="https://ajax.googleapis.com/ajax/
libs/jquery/1.5.1/jquery.min.js"></script>
```

Document Object Model (DOM)

- The Document Object Model (DOM) is a cross-platform and language-independent convention for representing HTML documents.
- The DOM is organized as a tree structure. Elements have ancestors and descendants, parents and children.



The jQuery Function

- The jQuery function is used to select elements from the DOM.
- It can either be used as `jQuery(<selector>)` or `$(<selector>)`.

```
jQuery(document.body).css('background', 'green');
```

```
$("#ul.menu").slideUp();
```

```
$('div.foo').click(function() {  
    $('p.bar img').hide();  
});
```

Selectors

- The jQuery library supports most CSS selectors.

`$(":radio:checked");`

`$(":button");`

form selectors

`$("a[href^=mailto:]");`

`$("a[href=.pdf]");`

atributte selectors

`$("td:contains(A)");`

`$("a:not(.foo)");`

custom selectors

- The `.filter()` method can be used to further restrict a selection.
- Methods can be chained to execute multiple actions in a single line.
- The DOM can be transversed using methods like `.parent()` or `.children()`.

`$("a.foo").parent();`

`$("p.bar").children();`

transversing

`$("a.foo").filter(".bar").children();`

`$("#bar").filter("p").first();`

filtering and chaining

Manipulation

- The jQuery `.css()` method can be used to get or set styles. There are several shortcuts, particularly for dimension properties, e.g. `.width()`, `.height()`.
- The attributes of any element may be get or set using the `.attr()` method.
- jQuery has methods to manipulate the classes of elements, e.g. `hasClass(<class>)`, `addClass(<class>)`, `removeClass(<class>)`, etc.
- The content of elements can be edited using the methods `.html()` or `.text()`.

```
$("p").css("color", "red");  
$("a.bar").attr("href");  
css and attributes
```

```
$("h1").addClass("active");  
$("#foo p").text("New text");  
classes and content
```

Manipulation Methods in a Nutshell

- To create new elements from HTML, use `$()`.
- To **insert** new elements **inside** every matched element, use `.append()`, `.appendTo()`, `prepend()`, `prependTo()`.
- To **insert** new elements **adjacent** to every matched element, use `.after()`, `.insertAfter()`, `.before()`, `.insertBefore()`.
- To **insert** new elements **around** every matched element, use `.wrap()`, `.wrapAll()`, `wrapInner()`.
- To **replace** every matched element with new elements, use `.html()`, `.text()`, `.replaceAll()`, `.replaceWith()`.
- To **remove** elements inside every matched element, use `.empty()`.

Events

- Tasks can be performed when the page loads or in response to some event.
- In jQuery `$(document).ready()` is invoked when the DOM is completely ready for use. Is the same as `$(()).ready()` of `jQuery(document).ready()`.
- It is possible to intercept user-initiated events by attaching event handlers to DOM elements. Some DOM events: click, dblclick, focus, keypress, mousemove, mouseover, resize, scroll, select, submit, etc.
- Functions can either be defined inline or defined elsewhere and invoked.

```
$(document).ready(function() {  
    // do something  
})
```

execute after page load

```
$("h1").click(function() {  
    // do something  
})
```

execute on given event

Event Propagation

- When an event occurs on a page, an entire hierarchy of DOM elements gets a chance to handle the event.
- Event propagation can be influenced to determine which elements get to respond to an event by using `.stopPropagation()` and `.preventDefault()`.
- The `.stopPropagation()` method halts the propagation process for the event.
- The `.preventDefault()` method will stop standard actions like following links or submitting forms on Enter.

```
$("p.foo").click(function(event) {  
    // do something  
    event.stopPropagation();  
}  
execute after page load
```

```
$("a").click(function(event) {  
    event.preventDefault();  
    // do something  
}  
execute on given event
```

Effects

- jQuery can be used to add simple visual effects and animations.
- The `.hide()` and `.show()` methods are frequently used for basic visual effects. Both methods accept a speed value that controls the duration of the effect.
- The `.animate()` method offers a way to create custom animations.

```
$("#bar").hide;  
$("#bar").show("fast");  
show and hide
```

```
$("#foo").animate({"left": "+=50px"}, "slow");  
$("#foo").animate({opacity: 0.25}, 5000);  
custom animations
```

- Several visual effects were separated from the core and organized in the jQuery UI library – <http://jqueryui.com/> (e.g. calendar widgets, sliders, etc).

jQuery Popular Plug-ins

- Form plugin – <http://malsup/com/jquery/form/>
- Autocomplete – <http://plugins.jquery.com/project/autocomplete>
- Validation – <http://plugins.jquery.com/project/validate>
- Tablesorter – <http://tablesorter.com/>
- FancyBox – <http://fancy.klade.lv/>
- Thickbox – <http://jquery.com/demo/thickbox/>
- Flot – <http://code.google.com/p/flot/>

jQuery Documentation

- **Selectors**

<http://api.jquery.com/category/selectors/>

- **Manipulation**

<http://api.jquery.com/category/manipulation/>

- **Effects**

<http://api.jquery.com/category/effects/>

- **Events**

<http://api.jquery.com/category/events/>

- **Plug-ins**

<http://plugins.jquery.com/>

Ajax

What is Ajax?

- Server calls from web pages using JavaScript



call
HTTP
data



Motivation

- The traditional request-response cycle in web applications is synchronous.
- With Ajax we can have asynchronous interactions, resulting in a much richer user experience.
- Principal motivation for Ajax use is the improved user experience in web applications.
- Make web applications more similar to desktop applications.

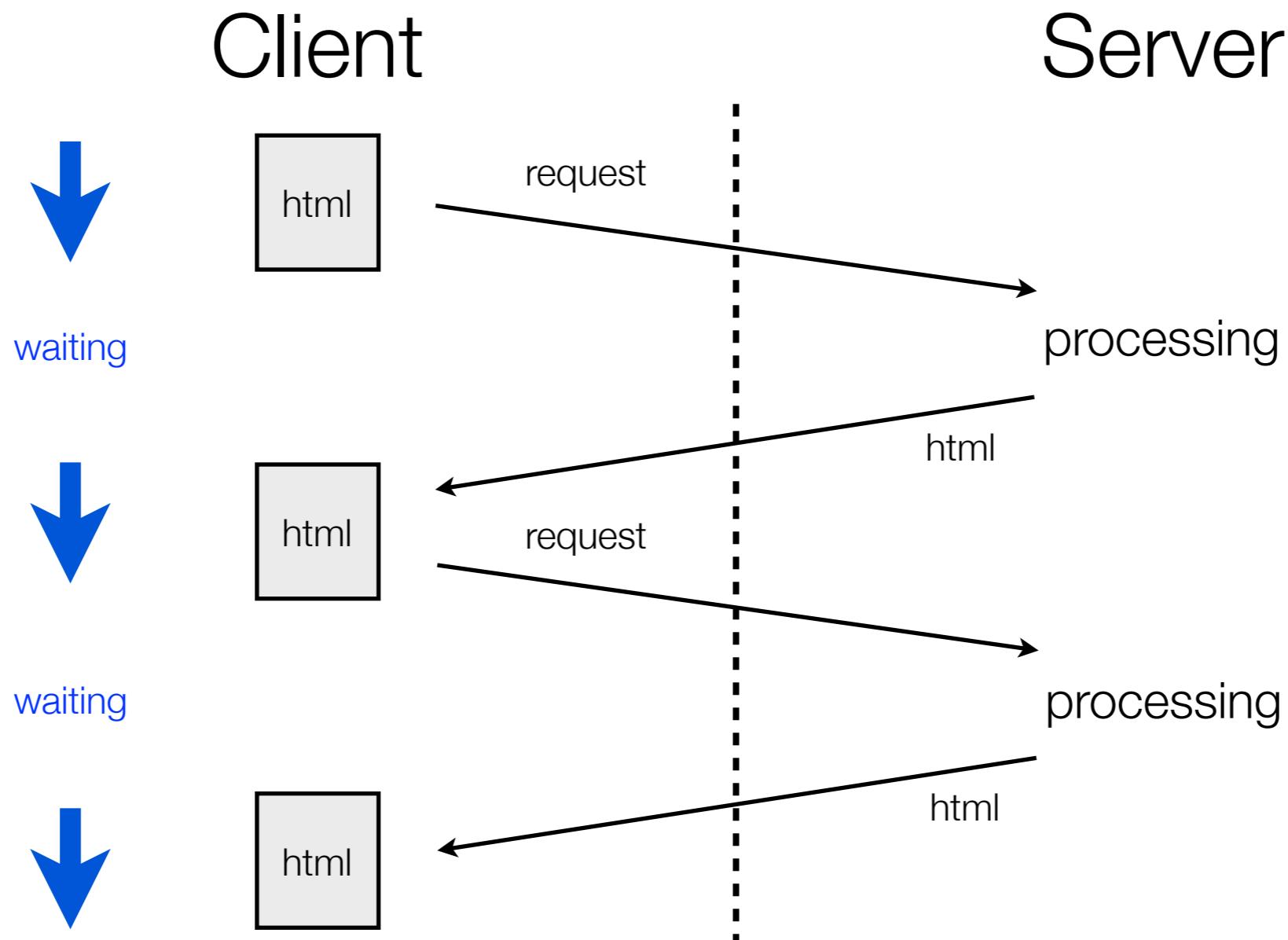
Brief History

- In 1999 Microsoft introduced the XMLHttpRequest feature in IE 5.
- Later adopted by other browser as the XMLHttpRequest JavaScript object.
- First web apps incorporating background HTTP requests date back to 2000.
- Wider visibility and adoption happened after GMail (2004) and Google Maps (2005).
- The term Ajax was coined in 2005.

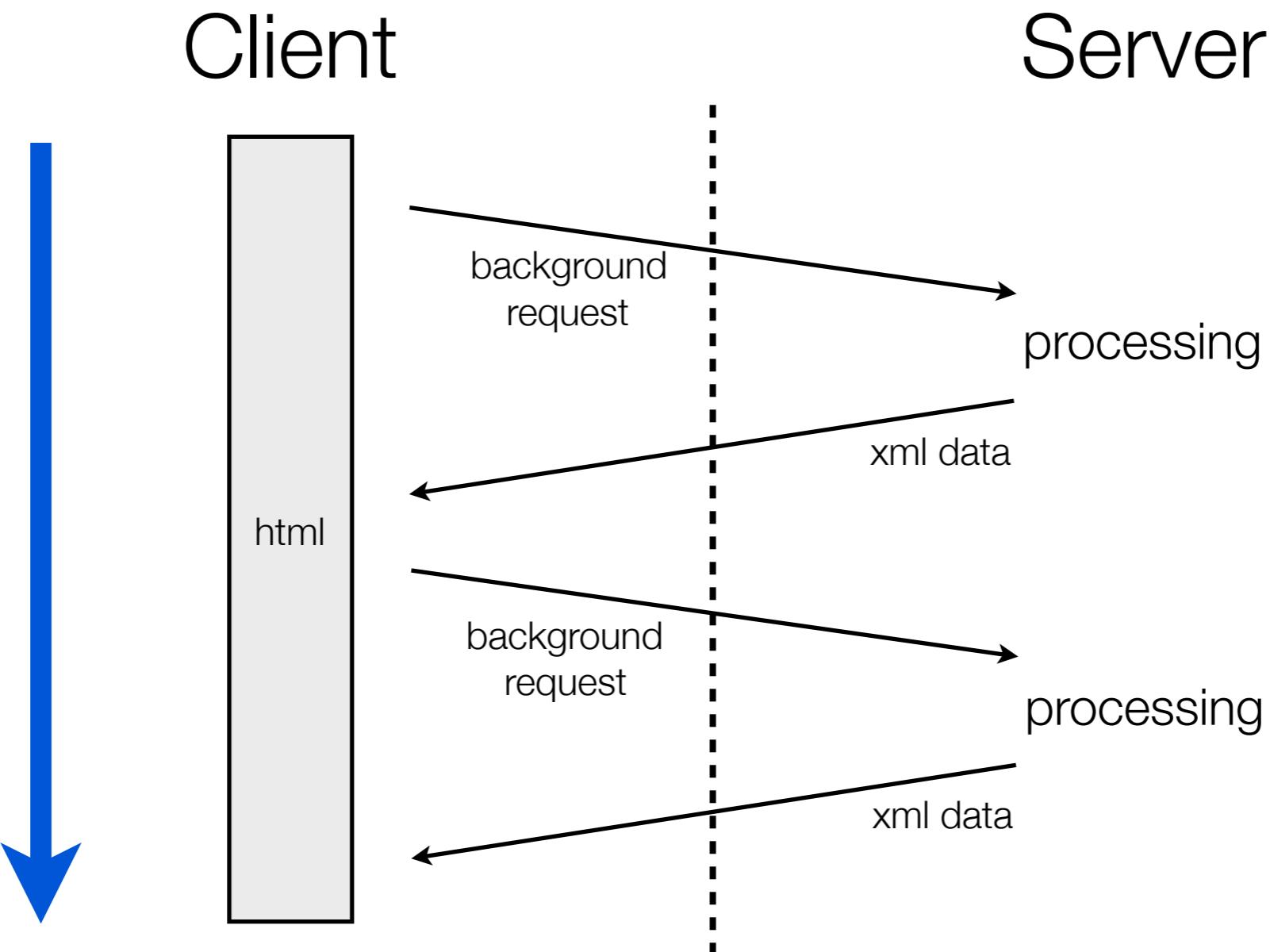
Ajax

- Ajax stands for Asynchronous JavaScript and XML.
- Ajax is a group of technologies combined to implement background calls in web applications.
- A browser technology independent of server software.
- Many libraries and toolkits are available to abstract low-level details (e.g. jQuery, Prototype, Dojo, etc).
- Typical use cases: form validation (e.g. mandatory fields), simple interactions (e.g. likes), input field suggestions.

Traditional Web App



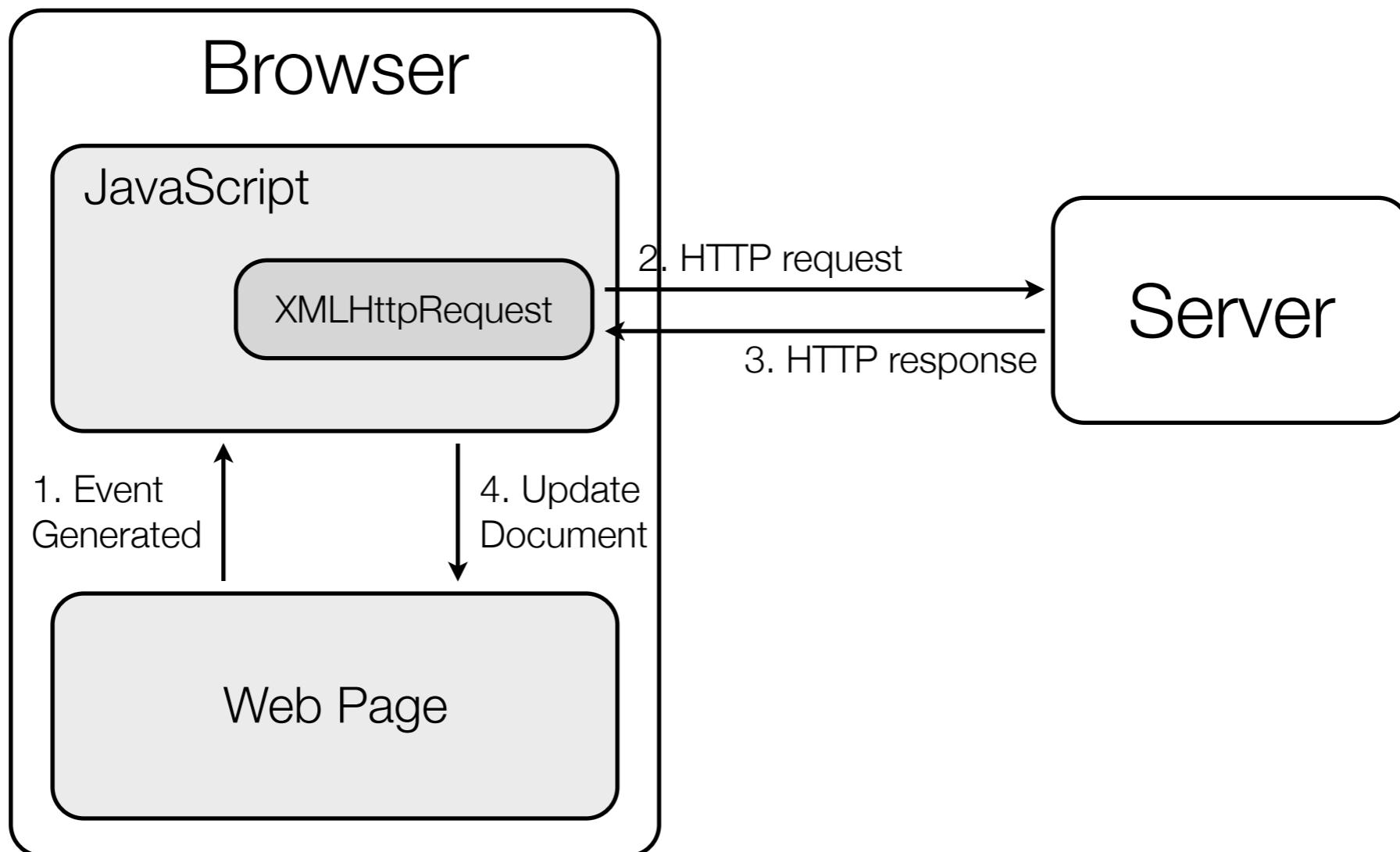
Ajax Web App



User interaction is independent of server access.

XMLHttpRequest object

- JavaScript object that can handle communication between clients and servers. The XMLHttpRequest object is created by the client.



XMLHttpRequest Example

```
var xmlhttp = new XMLHttpRequest();
```

Create XMLHttpRequest object.

Not so simple in real world applications. Older browser versions need to be supported.

```
xmlhttp.open("GET", "api/search.php", true);  
xmlhttp.onreadystatechange = handleResponse;
```

Make asynchronous call, limited to same domain.

Define function to handle response.

```
function handleRequest() {  
  if (xmlhttp.readyState == 4) {  
    // ok ...  
  } else { // not ok ... }  
}
```

Check response code.

Response body is available in xmlhttp.responseXML.

JSON

- JSON stands for JavaScript Object Notation.
- A simple text-based, lightweight data interchange format that is easy to generate and easy to parse.
- Smaller footprint than XML, thus higher transmission speeds.

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
</person>
```

XML

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25
}
```

JSON

JSON Examples

```
{  
  "firstName": "John"  
}
```

Object with one member.
`var["firstName"] // John`

```
{  
  "firstName": "John",  
  "lastName": "Smith"  
}
```

Object with two members.
`var["lastName"] // Smith`

```
{  
  "firstName": "John",  
  "tags": ["a", "b", 1]  
}
```

Object with array as value.
`var["tags"][0] // a`

```
{  
  "firstName": "John",  
  "address": {  
    "street": "Sesame",  
    "city": "NYC"  
  }  
}
```

Object with objects.
`var["address"]["street"] // Sesame`

Ajax with jQuery

```
jquery.ajax( url, [ settings ] )
```

url – URL to which the request is sent.

settings – key/value pairs that configure the Ajax request

- async: if false, the request is made synchronously.
- data: data to be sent to the server, as key/value pair.
- type: the type of HTTP request to be made (e.g. POST/GET).
- cache: if false will force the browser not to use cache.
- success: a function to be called on success.

...

```
$.ajax({  
    type: "GET",  
    url: "test.js",  
    dataType: "script"  
});
```

Load and execute a JavaScript file.

```
$.ajax({  
    type: "POST",  
    url: "some.php",  
    data: "name=John&location=Boston",  
    success: function(msg){  
        alert( "Data Saved: " + msg );  
    }  
});
```

Save data to server and notify user when complete.

```
$.ajax({  
    url: "test.html",  
    cache: false,  
    success: function(html){  
        $("#results").append(html);  
    }  
});
```

Retrieve the latest version of an HTML page.

Parsing JSON

- jQuery.parseJSON(json)

Takes a well-formed JSON string and returns the resulting JavaScript object.

```
var obj = jQuery.parseJSON('{"name":"John"}');
alert( obj.name === "John" );
```

Cross-domain Ajax

- The same-origin policy prevents scripts from one domain to access or manipulate properties or documents from another domain.
- This policy limits standard Ajax calls to the same server.
- How to make cross-domain Ajax calls? (e.g. use external APIs)
- Workaround: JSONP or JSON with Padding.
- The HTML script element, not limited by the same-origin policy, is used to load external data and call a local function defined in the callback parameter.

```
<script type="text/javascript"  
src="http://example.com/api/foo.json?callback=processData">  
</script>
```

1. Make call within the script element, include the local function name for the callback.

```
processData({"id":123, "name":"foo"})
```

2. Server sends data padded with function name.

3. The local function is called with the remote data

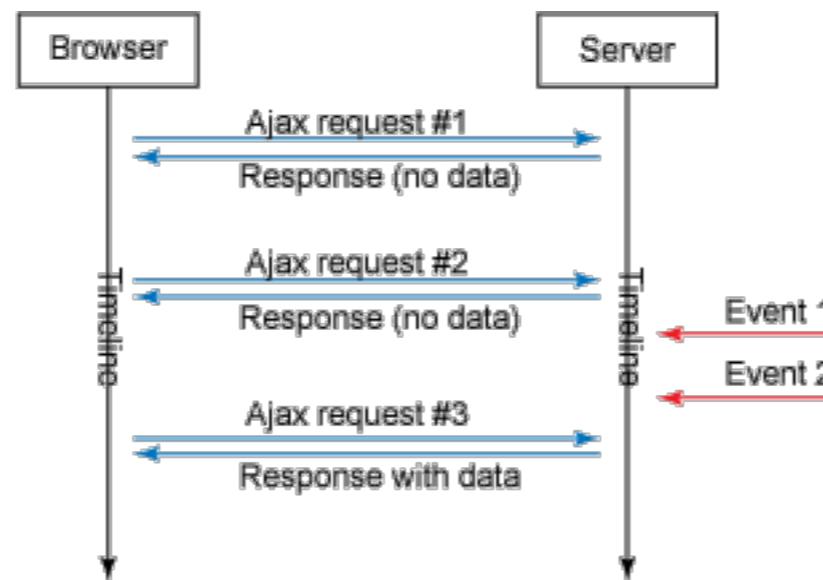
Reverse AJAX

Reverse Ajax

- Ajax calls are initiated by the client.
- How to push information from the server to the client?
- Use cases: notifications on long running tasks, chat systems, multi-user collaboration systems (e.g. live collaborative text editors).

Polling

- Make periodic requests to the server to check for new data.

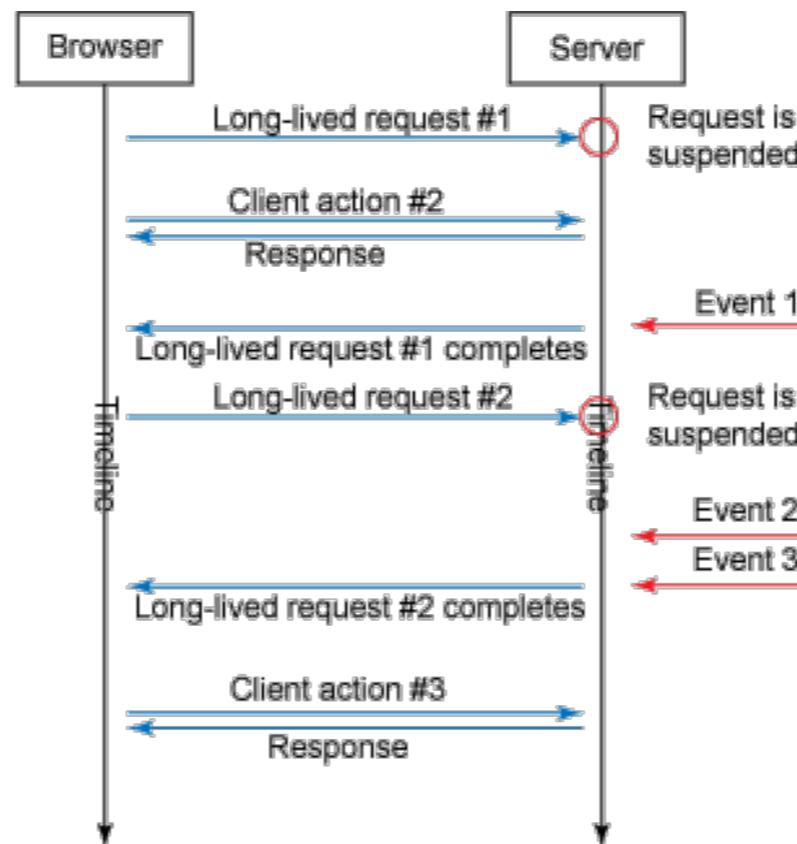


- The smaller the interval between requests the more up to date the data is.
- Drawbacks: resource and bandwidth consumption even when no new data is available. Does not scale well and doesn't guarantee low-latency.

Image source: <http://www.ibm.com/developerworks/web/library/wa-reverseajax1/>

Comet

- Requests are initiated by the clients and kept alive for long periods, until a time-out occurs or a response is sent.

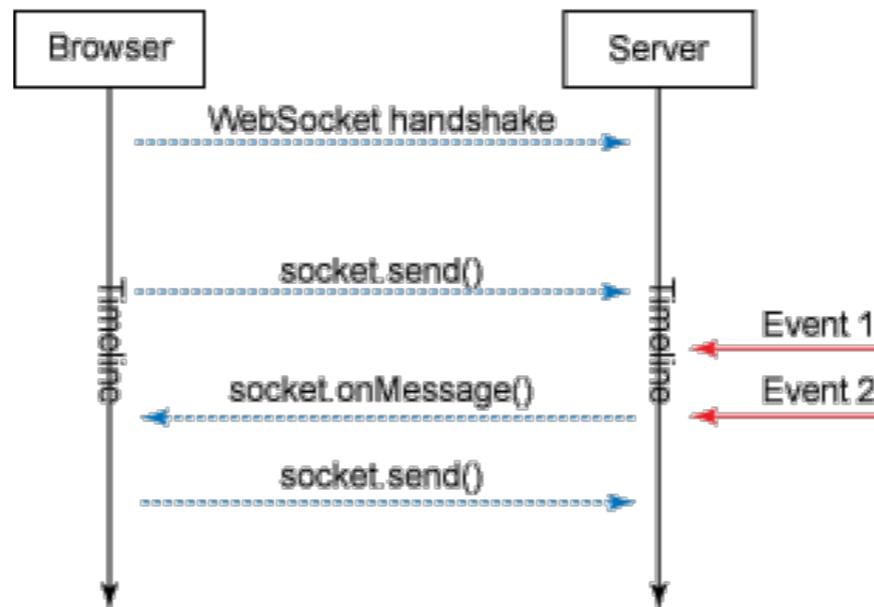


- On the server, the request is suspended or paused until a response is ready.

Image source: <http://www.ibm.com/developerworks/web/library/wa-reverseajax1/>

Web Sockets

- Web Sockets is an API that enables bidirectional communications between the web browser and the web server. No polling is needed to get messages from the server.



- Browser support: <http://caniuse.com/websockets>
- PHP support: <http://socketo.me/>

Image source: <https://www.ibm.com/developerworks/library/wa-reverseajax2/>

Ajax Drawbacks

- Pages dynamically created using Ajax requests are not available in the browser's history.
- Dynamic web page updates make it difficult to bookmark particular state of an application.
- Dynamically created web pages are inaccessible to web crawlers, given that they require JavaScript execution.
- Reduced accessibility to clients not supporting JavaScript. Also limited support in screen-readers.
- May lead to complex code, harder to maintain or debug.

Client-Side JavaScript Frameworks

Client-Side JavaScript Frameworks

- Modern web applications have an important part of their logic in JavaScript.
- JavaScript Frameworks have emerged to help manage this growing codebase. Advantages: modularity, maintainability, productivity.
- Client-side web applications are an important trend.
- Important limitations: search engine indexing, caching (REST), client-side memory / performance, latency (e.g. pages with many database calls).

Popular Frameworks

- **React**

<https://reactjs.org/>

- **Vue.js**

<https://vuejs.org/>

- **AngularJS**

<http://angularjs.org/>

- Same project using different frameworks – <http://todomvc.com/>

References

- Learning jQuery 1.3
Jonathan Chaffer, Karl Swedberg. Packt (2009)
- jQuery Cookbook
jQuery Community Experts, O'Reilly (2010)
- JavaScript Guide - MDC Doc Center
<https://developer.mozilla.org/en/JavaScript/Guide>
- A re-introduction to JavaScript - MDC Doc Center
[https://developer.mozilla.org/en/A re-introduction to JavaScript](https://developer.mozilla.org/en/A_re-introduction_to_JavaScript)
- Reverse Ajax, Part 1: Introduction to Comet - IBM Developer Works
<http://www.ibm.com/developerworks/web/library/wa-reverseajax1/>