JÖNKÖPING UNIVERSITY

School of Engineering

AJAX & JAVASCRIPT LIBRARIES

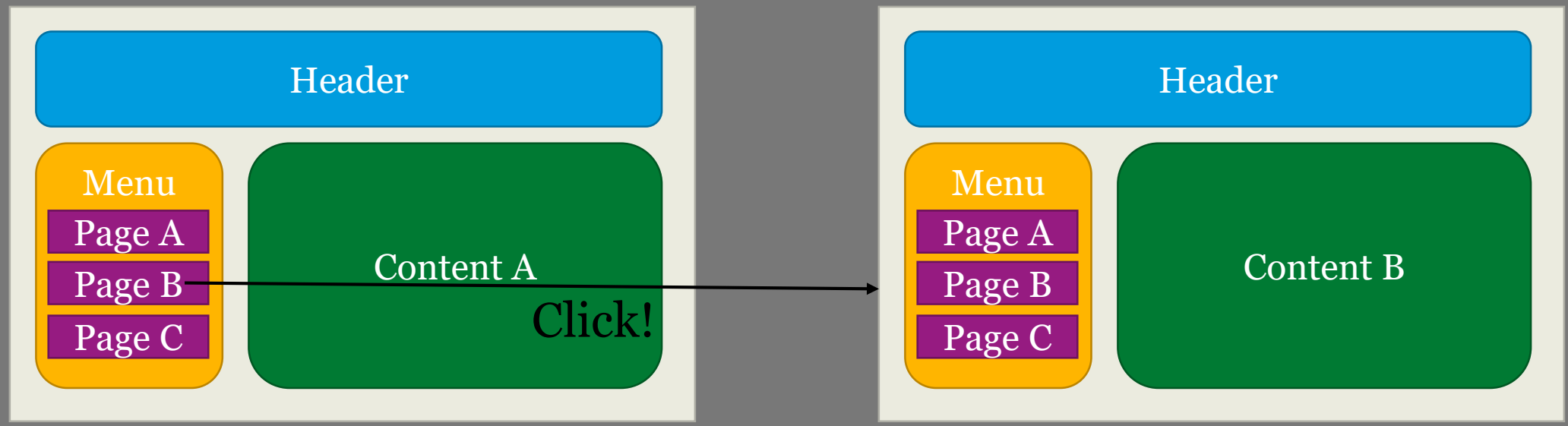
Web Development with JavaScript and DOM

TWJK14 Spring 2017

Peter Larsson-Green

JÖNKÖPING UNIVERSITY
School of Engineering

TRADITIONAL WEB PAGES



`page-a.html` `page-b.html`

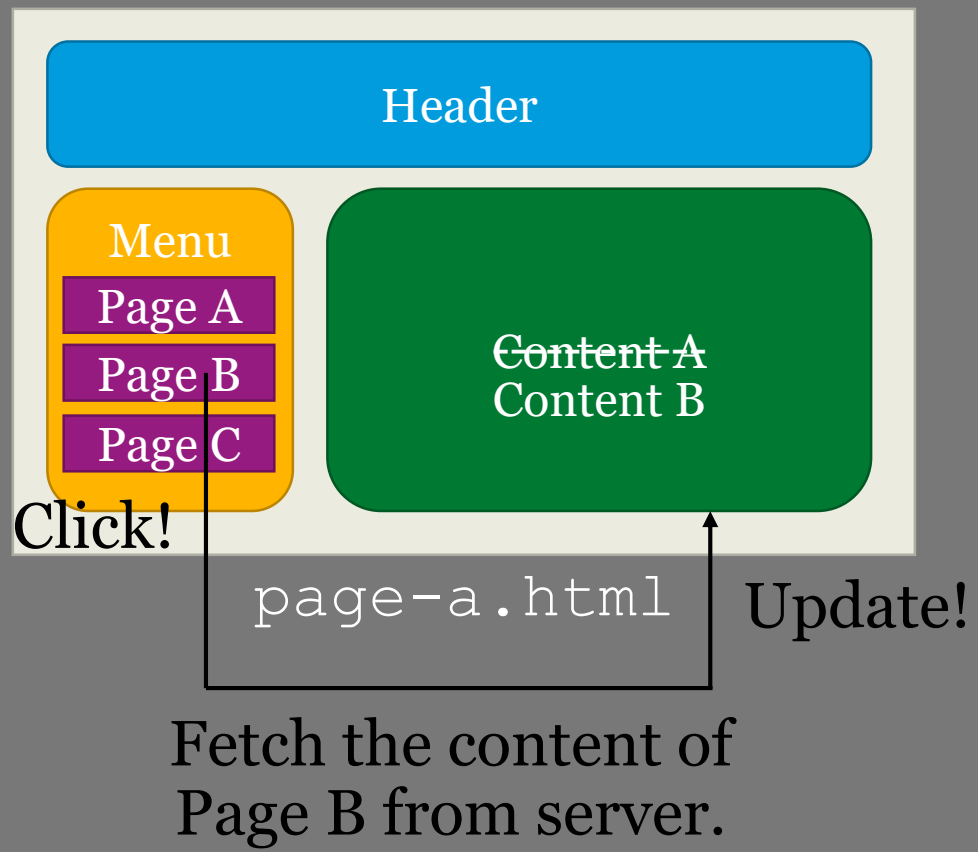
We fetch information we already have.

- Takes extra time ☹
- Wasting data usage ☹

The browser re-render the same thing over again.

- The screen flickers ☹

MODERN WEB PAGES



When we fetch the content for Page B, we usually just get back data.

- No HTML code.
- No CSS code.
- No JavaScript code.

The data can exist in different formats.

- XML.
- JSON.
- …

XML

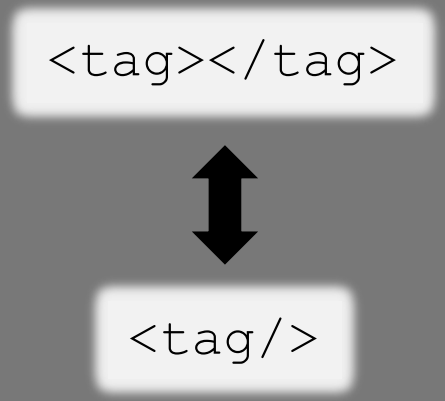
A simplified view!

eXtensible Markup Language

Specification: https://www.w3.org/TR/xml

Declaration

Root




```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book isbn="0-671-62964-6">
    <title>The Hitchhiker's Guide to the Galaxy</title>
    <author>Douglas Adam</author>
  </book>
  <book isbn="0-7475-3269-9">
    <title>Harry Potter and the Philosopher's Stone</title>
    <author>J. K. Rowling</author>
  </book>
</books>
```

AJAX

Asynchronous JavaScript and XML

How do we fetch new content from the server?

- JavaScript does not contain that functionality.

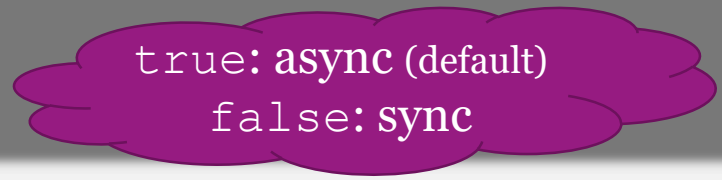
BOM includes the `XMLHttpRequest` object.

- Use it to send HTTP requests to servers and receive HTTP responses.
- Is not limited to XML files.
- Also known as XHR.

THE XMLHTTPREQUEST OBJECT

Specification: https://xhr.spec.whatwg.org/

Sending a request:




```
var request = new XMLHttpRequest()
request.open("METHOD", "the-url", true)
request.setRequestHeader("Name-Of-Header", "Value")
request.send("The body of the request.")
```

Listening for the response (if used asynchronously):

```
request.onreadystatechange = function(){
  // I am called each time request.readyState changes.
}
```

THE XMLHTTPREQUEST OBJECT

Listening for the response (if used asynchronously):


```
request.onreadystatechange = function(){
  if(request.readyState == XMLHttpRequest.UNSENT){
    // The open method hasn't been called.
  }else if(request.readyState == XMLHttpRequest.OPENED){
    // The open method has been called.
  }else if(request.readyState == XMLHttpR
    // Response headers have been receive
  }else if(request.readyState == XMLHttpRequest.LOADING){
    // Is receiving the body now.
  }else if(request.readyState == XMLHttpRequest.DONE){
    // The entire response has been received.
  }
}
```

```
request.onError = function(){
  // Deal with errors.
}
```

THE XMLHTTPREQUEST OBJECT

Listening for the response (if used asynchronously):


```
request.addEventListener("load", function(){
  // The response has been received.
})
```

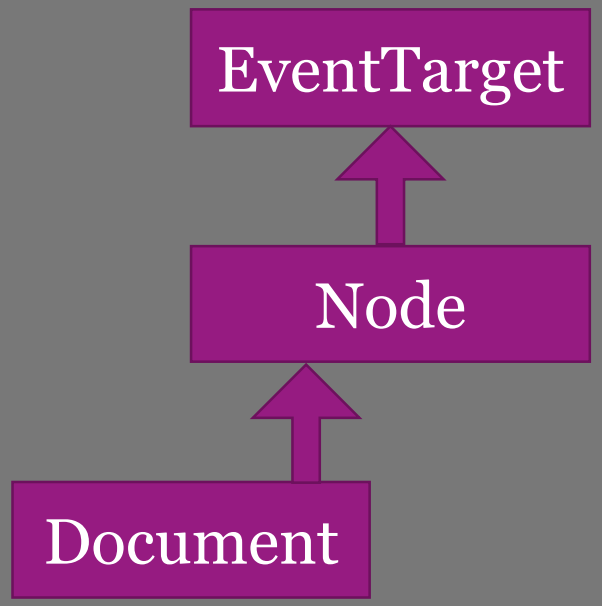
```
request.addEventListener('error', function(){
  // Deal with errors.
}
```

Reading the response:

```
var statusCode = request.status
var aHeader = request.getResponseHeader("Name-Of-Header")
var bodyAsString = request.responseText
var bodyAsXmlDocument = request.responseXML
```

XML DOCUMENT

```
var bodyAsXmlDocument = request.responseXML
```



EXAMPLE

```
var request = new XMLHttpRequest()
request.open("GET", "letters.xml")
request.setRequestHeader("Accept", "application/xml")
request.addEventListener("load", function(){
  var xmlDoc = request.responseXML
  var letterElements = xmlDoc.getElementsByTagName("letter")
  for(var i=0; i<letterElements.length; i++){
    var letterElement = letterElements[i]
    var letter = letterElement.textContent
    console.log(letter)
  }
})
request.send()
```

```
<?xml version="1.0"?>
<letters>
  <letter>a</letter>
  <letter>b</letter>
  <letter>c</letter>
</letters>
```

SECURITY ISSUE #1

If `XMLHttpRequest` could request local files,
any website on the Internet could access any file on your computer.

- Browsers do not allow this using the HTTP protocol.
- Some browsers even disallow it over the `file` protocol.
  - Firefox allows it.
  - Microsoft Edge allows it.
  - Start Google Chrome with the flag `--allow-file-access-from-files`
    - In Windows PowerShell: `Start-Process "chrome.exe" "--allow-file-access-from-files"`

JSON

XML contains a lot of characters → JSON invented.

```
<?xml version="1.0"?>
<letters>
  <letter>a</letter>
  <letter>b</letter>
  <letter>c</letter>
</letters>
```

```
{"letters": [
  {"letter": "a"},
  {"letter": "b"},
  {"letter": "c"},
]}
```

```
["a", "b", "c"]
```

JSON

JavaScript Object Notation

- JSON code consists of one of the following literal expressions:
  - String: `"Hi!"`
  - Number: `12.345`
  - Booleans: `true`
  - Null: `null`
  - Object: `{"key-a", value, "key-b", value, <expr>, ...}`
  - Array: `[value, value, ...]`
- Specification: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
- Easy-to-understand specification: http://www.json.org

JSON

JavaScript Object Notation

- The `JSON` object contains useful functions:
  - `JSON.stringify({a: 1, b: 2})` → `'{"a": 1, "b": 2}'`
  - `JSON.parse('{"a": 1, "b": 2}')` → `{a: 1, b: 2}`

EXAMPLE

```
var request = new XMLHttpRequest()
request.open("GET", "letters.json")
request.setRequestHeader("Accept", "application/json")
request.addEventListener("load", function(){
  var body = request.responseText
  var letters = JSON.parse(body)
  for(var i=0; i<letters.length; i++){
    var letter = letters[i]
    console.log(letter)
  }
})
request.send()
```

```
["a", "b", "c"]
```

SECURITY ISSUE #2

If `XMLHttpRequest` could send requests to other domains, it could access any website on the behalf of the user.

Example

1. A user visits `bank.com`.
2. The user signs in (and never signs out).
3. The user then goes to `evil-site.com` (by mistake).
4. The evil site use `XMLHttpRequest` to send HTTP POST requests to `bank.com` transferring money from the user's account.

Browsers do not allow this.

- Known as *the same-origin policy*: `XMLHttpRequest` can only be used to send requests to the same domain the webpage comes from.

JSONP

An attempt to avoid the same-origin policy.

- The `<script>` element is not limited by the same-origin policy.

```
<script src="http://other-domain.com/the-file.js"></script>
```

  - Use `<script>` to send request to other domains!

```
var script = document.createElement("script")
script.src = "http://other-domain.com/the-file.js"
document.head.appendChild(script)
```

The browser will fetch the JavaScript file and then execute the code in it.

JSONP

An attempt to avoid the same-origin policy.

- How do we pass information to the server?
  - In the query string!

```
var script = document.createElement("script")
script.src = "http://other-domain.com/the-file.js"
script.src += "?parameter-a=value-a"
script.src += "&parameter-b=value-b"
document.head.appendChild(script)
```

JSONP

An attempt to avoid the same-origin policy.

- How do we read information back from the server?
  - The server returns JavaScript code the browser execute.
    - The code consists of a call to a function you create.
    - Information is passed as arguments.

Many web servers let you specify the name of this function in the query string.

```
var script = document.createElement("script")
script.src = "http://other-domain.com/get-books-as-jsonp"
document.head.appendChild(script)
function hereAreTheBooks(books){
  // Called when we
  // receive the books.
}
```

```
hereAreTheBooks([
  {title: "Book A", pages: 152},
  {title: "Book B", pages: 384}
])
```

JSONP

An attempt to avoid the same-origin policy.

- Why is it called JSONP?
  - JSON with Padding

JSON:

```
{"a": 1, "b": 2}
```

JSONP:

```
callback({"a": 1, "b": 2})
```

Padding

JSONP

Try it yourself:

- Free JSONP available:
  - http://www.icndb.com/api/

CORS

Cross-Origin Resource Sharing.

- A protocol developed for secure cross-origin requests.
- The same-origin policy can be avoided if:
  - The web server supports CORS.
  - The client supports CORS.
- How it works (partially):
  1. An `XMLHttpRequest` wants to send a request to another-domain.com.
  2. The browser sends another request to another-domain.com, asking if the server allows cross-origin requests.
  3. If the server allows it, the client sends the `XMLHttpRequest` to the server.
- Learn more: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

FETCH

The new way to send HTTP request from the browser.

- Not supported in all browsers at the moment:
  - http://caniuse.com/#feat=fetch
- Biggest improvement to `XMLHttpRequest`: promise instead of `addEvenetListener`.

BROWSER SUPPORT VARIES

Browser support for HTML & JavaScript varies.

- Using CORS:
  - Modern browsers use `XMLHttpRequest` object.
  - IE8 & IE9 use `XDomainRequest` object.
- Listening for events:
  - Modern browsers use `addEventListener`.
  - IE6-8 use `attachEvent`.
- Using `getElementsByClassName`:
  - IE6-8 do not support it.

BROWSER SUPPORT VARIES

Browser support for HTML & JavaScript varies.

- Using `classList`:
  - IE <= 9 do not support it.
  - IE10 & IE11 support parts of it.
- The `DOMContentLoaded` event:
  - IE <= 8 do not support it.
- Using form validation attributes:
  - IE <= 9 do not support it.

BROWSER SUPPORT VARIES

What can we do?

- Features we cannot do anything about:
  - Give alternative solution: Instead of playing the song, show the lyrics.
- Features with different names:
  - Write a lot of code.

ADDING A CLICK LISTENER

```
var p = document.getElementById("p")
if(p.addEventListener != undefined){
  p.addEventListener('click', function(event){
    // Handle it!
  })
}else if(p.attachEvent != undefined){
  p.attachEvent('onclick', function(){
    var event = window.event
    // Handle it!
  })
}else{
}
```

```
var oldOnclick = p.onclick
p.onclick = function(event){
  event = event || window.event
  // Handle it!
  oldOnclick && oldOnclick()
}
```

BROWSER SUPPORT VARIES

What can we do?

- Features we cannot do anything about:
  - Give alternative solutions: Instead of playing the song, show the lyrics.
- Features with different names:
  - Write a lot of code.
  - Preferably: use a library.
- Some missing features:
  - Simulate them using HTML, CSS and JavaScript.

SIMULATING MISSING FEATURE

```
<form id="searchForm">
  Search: <input type="search" required id="searchInput">
          <input type="submit" value="Send">
</form>
```

```
var form = document.getElementById("searchForm")
form.addEventListener('submit', function(event){
  var input = document.getElementById("searchInput")
  if(input.value == ""){
    event.preventDefault()
    alert("You must enter a search term.")
  }
})
```

JQUERY

A popular JavaScript library.

- Use library → we can be sure our code works the same across browsers.
- Webpage: http://jquery.com

VANILLA JAVASCRIPT VS JQUERY

```
document.addEventListener("DOMContentLoaded", function(){
  // Not in IE <= 8.
})
```

```
$(document).ready(function(){
  // Handle it.
})
```

```
$(function(){
  // Handle it.
})
```

VANILLA JAVASCRIPT VS JQUERY

```
var a = document.getElementById("id")
var b = document.getElementsByTagName("tag")
var c = document.getElementsByClassName("class") // Not in IE <= 8.
var d = document.querySelector("cssSelector") // Not in IE <= 7.
var e = document.querySelectorAll("cssSelector") // Not in IE <= 7.
```

```
var allMatches = $("cssSelector") // A jQuery array-like object.
```

VANILLA JAVASCRIPT VS JQUERY

```
theElement.classList.add("newClass") // Not in IE <= 9.
theElement.className += " newClass"
theElement.setAttribute(
  "class",
  theElement.getAttribute("class") + " newClass"
)
```

```
jQueryObject.addClass("newClass")
```

VANILLA JAVASCRIPT VS JQUERY

```
theElement.addEventListener("click", function(event){
  // Not in IE <= 8.
})
```

```
jQueryObject.on("click", function(event){
  // Handle it!
})
```

VANILLA JAVASCRIPT VS JQUERY

```
var request = new XMLHttpRequest()
request.open("GET", "path/to/the/file.json")
request.setRequestHeader("Content-Type", "application/json")
request.setRequestHeader("Accept", "application/json")
request.addEventListener("load", function(){ }) // Not in IE <= 8.
request.send('{"the": "data"}')
```

```
$.ajax({
  method: "GET"
  url: "path/to/the/file.json",
  contentType: "application/json",
  dataFormat: "json",
  success: function(response){ },
  data: '{"the": "data"}'
})
```

VANILLA JAVASCRIPT VS JQUERY

```
var script = document.createElement("script")
script.src = "path/to/jsonp/file?callback=theCallback"
function theCallback(data){ }
document.head.appendChild(script) // Not in IE <= 8.
```

```
$.ajax({
  url: "path/to/jsonp/file",
  datatype: "jsonp"
  success: function(response){ }
})
```

DETECTING AVAILABLE FEATURES

```
var input = document.createElement("input")
input.setAttribute("type", "number")
if(input.type != "text"){
  // <input type="number"> is supported!
}
```

```
if("localStorage" in window){
  // localStorage is supported!
}
```

```
if(window.localStorage){
  // localStorage is supported!
}
```

MODERNIZR

A library for detecting available features.

- Webpage: https://modernizr.com

```
if(Modernizr.inputtypes.date){
  // <input type="date"> is supported!
}
```

POLYFILLS

Code simulating missing features.

- Examples:
  - `localStorage` not supported?

```
var localStorage = {
  setItem: function(key, value){ /* Store in a cookie. */ },
  getItem: function(key){ /* Get from cookie. */ },
  // ...
}
```

POLYFILLS

Code simulating missing features.

- Examples:
  - `localStorage` not supported?
  - HTML5 validation attributes not supported?

```
var forms = document.querySelectorAll('form')
for(var i=0; i<forms.length; i++){
  forms[i].addEventListener('submit', function(e){
    var inputs = e.currentTarget.querySelectorAll('input')
    // Check if any input has any HTML 5 validation
    // attribute, and if so use JavaScript to validate it.
  })
}
```

POLYFILLS

Code simulating missing features.

- Examples:
  - `localStorage` not supported?
  - HTML5 validation attributes not supported?
- List of some polyfills: https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills

```
if(!Modernizr.placeholder){
  var script = document.createElement("script")
  script.src = "path/to/polyfill/for/placeholder.js"
  document.head.appendChild(script)
}
```

RECOMMENDED READING

Specifications:

- XMLHttpRequest:
  - https://xhr.spec.whatwg.org/
- JSON:
  - https://tools.ietf.org/html/rfc7159
  - http://www.json.org/
- XML:
  - https://www.w3.org/TR/REC-xml/

RECOMMENDED READING

W3Schools:

- XML Tutorial:
  - https://www.w3schools.com/xml/default.asp
- JSON Introduction:
  - https://www.w3schools.com/js/js_json_intro.asp
- AJAX Introduction:
  - https://www.w3schools.com/xml/ajax_intro.asp
  - https://www.w3schools.com/js/js_ajax_intro.asp
- jQuery Tutorial:
  - https://www.w3schools.com/jquery/