TEMA 4. PROGRAMACIÓN DE APLICACIONES EN EL CLIENTE

Curso 2018-2019

Tecnologías Web

Bibliografía

- R. Nixon, «PHP, MySQL, & Javascript», O'Reilly, 2009
- T. Wright, «Learning JavaScript», Addison-Wesley, 2013
- D. Flanagan, «JavaScript: The Definitive Guide», 5th Edition, O'Reilly, 2006
- A. Freeman «Pro JavaScript Web Apps», Apress, 2012

Contenido

- Aplicaciones web en el cliente
- Lenguaje JavaScript
- JavaScript Avanzado
- AJAX

APLICACIONES WEB EN EL CLIENTE

Aplicación web en el cliente

- Ejecución de aplicaciones en el dispositivo cliente (navegador)
- Poca carga computacional
- Más interactividad y más rápida
  - Validación sintáctica/semántica de entradas
  - Información contextual
  - Efectos de animación
  - Gráficos interactivos
  - …

El navegador se está convirtiendo en la nueva plataforma nativa

Requisitos

- Lenguaje multiplataforma
- ¿Interpretado?
- Impone carga leve de traducción y ejecución
- Fácilmente integrable con el lenguaje de marcado (HTML)

LENGUAJE JAVASCRIPT

Lenguaje JavaScript

- Lenguaje de alto nivel, débilmente tipado, incrustable en HTML, con soporte para PDO, interpretado y ejecutado en el navegador
- Tiene acceso a todos los elementos del documento web
- Aparece en 1995 con el navegador Netscape
- Nombre oficial: EMACScript
- No tiene ninguna relación directa con Java

Modos de uso

1. En línea, asociado a eventos:

```
<a href="/about" onclick=
“alert(‘mensaje');” > About</a>
```

2. Incrustado: `<script> … </script>`

3. En ficheros externos:

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

El fichero no puede incluir <script>

Script sencillo

hola-js.html

```
<script type=“text/javascript”>
  window.alert("Hola mundo");
</script>
```

Ejemplos en http://betatun.ugr.es/~jmbs

Ubicaciones en el documento

- Cabecera:

  Procesado antes de recibir el cuerpo del documento; disponible en todo el documento.

  Escribir metadatos
- Cuerpo
- Externo

Depuración de JavaScript

- Los mensajes no se muestran en la página
- Depende del navegador. No hay uniformidad
- Habitualmente existe una «Consola de errores» donde se muestran los mensajes

Elementos del lenguaje JavaScript

- Sentencias; Comentarios
- Variables
- Operadores
- Tipos de datos; *arrays*
- Concatenadores
- Funciones
- Expresiones y control de flujo
- PDO
- Formularios

Sentencias

- El intérprete se activa para bloques delimitados por:

  <script> …</script>

- No es necesario que las sentencias terminen con «;», aunque sí es necesario para separar sentencias
- Sintaxis que recuerda a la de C/C++, Perl, …

Variables en JavaScript

- No hay verificación de tipos. Una variable es un espacio para almacenar datos

- Nombres formados por letras, números o $
- No tienen que comenzar por $
- Distingue entre Mayúsculas y minúsculas

Operadores

- Aritméticos: +, -, *, /, %, ++, --
- De asignación: =, +=, -=, *=, /=, %=, .=
- Lógicos: &&, ||, !
- Comparación: ==, !=, <, >, <=, >=, ===, !==
- De bit: !,
- Concatenación de cadenas: +
- Casting: (int), (double), (string), (array), (object)

Tipos de datos

- Numéricos: enteros, coma flotante
- Lógicos: booleanos
- Cadenas de caracteres:
  - Literales: encerradas entre comillas simples
  - Interpretadas: encerradas entre comillas dobles

Arrays

- Tipo de dato estructurado (no necesariamente homogéneo)
- Se crean con `Array()` o simplemente `[]`
- Los componentes se acceden mediante índices, comenzando en 0. También los hay asociativos (diccionarios)
- Pueden ser multidimensionales:

```
matriz = Array(Array(1, 2, 3), Array(4, 5, 6))
```

- Crecen mediante el método `push`.

Funciones

```
<script>
function suma(a, b)
{
     return a + b
}
</script>
```

Argumentos: nombre_funcion.arguments

Extensa disponibilidad de **bibliotecas** de funciones en JavaScript

Ámbito de las variables

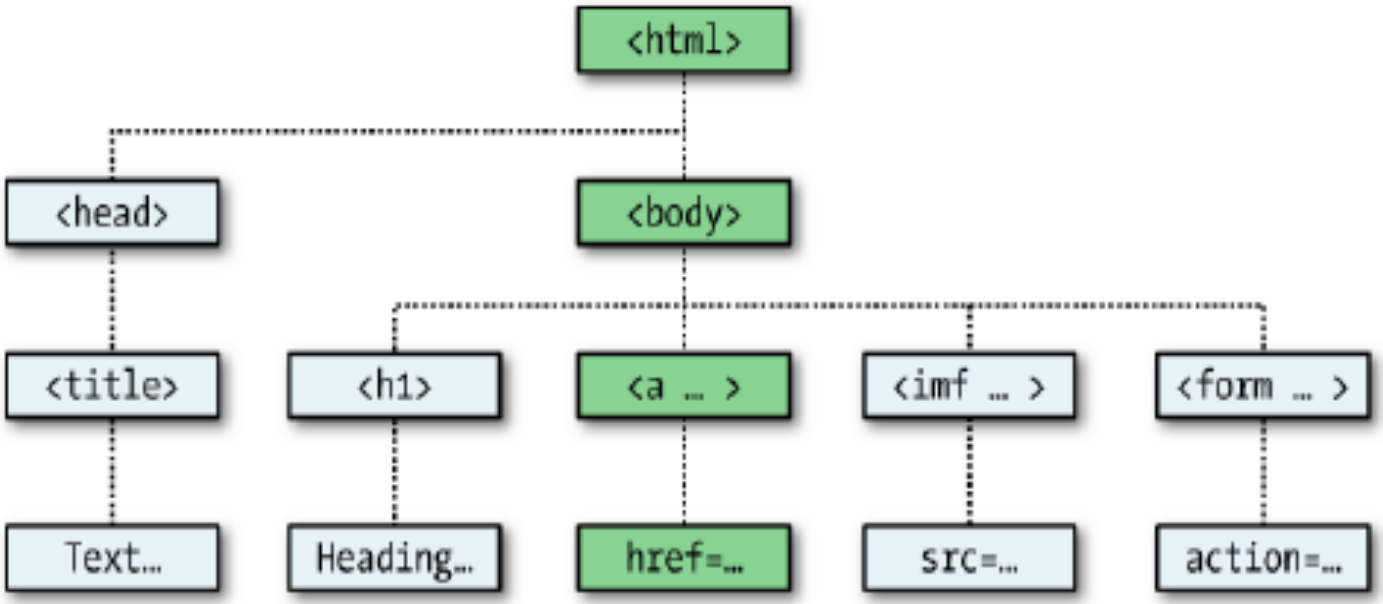
- Local: en el ámbito de la función en que se definen
- Global: definidas fuera de una función

Document Object Model (DOM)

- JavaScript está diseñado entorno al Modelo de Objetos Documentales (DOM): Las partes de un documento HTML son objetos, cada uno con sus variables de instancia y métodos.
- Notación para acceso a miembros: «.»
- La relación entre los objetos es jerárquica:

```
url = document.links.linkname.href
```

Jerarquía de objetos



jerarquia-js.html

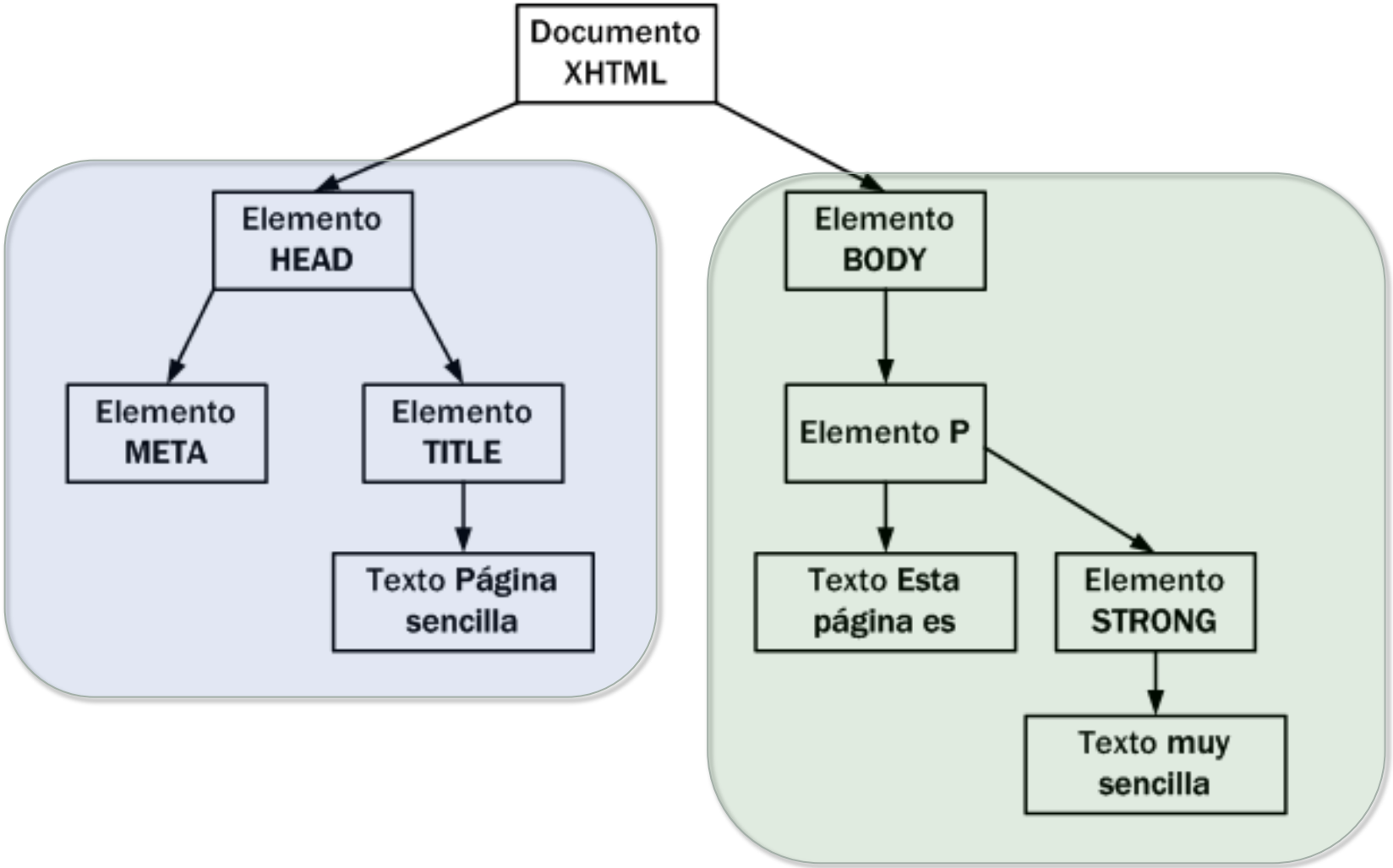
```
<html>
<head>
      <title>Link Test</title>
</head>
<body>
      <a id="mylink" href="http://betatun.ugr.es">
            Click me</a><br />
      <script>
            url = document.links.mylink.href
            document.write('El URL es ' + url)
      </script>
</body>
</html>
```

arbol-nodos.html

```
<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Página sencilla</title>
</head>

<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Árbol (DOM) del arbol-nodos.html



DOM: Tipos de nodos

- Document
- Element
- Attr
- Text
- Comment
- Otros 7

Arrays de objetos

- En realidad, cada objeto es un array:

```
url = document.links[0].href
```

- Acceso a todos los enlaces de un documento:

```
for (j=0 ; j < document.links.length ; ++j)
  document.write(document.links[j].href + '<br />')
```

PDO

- El paradigma PDO está incluido en Javascript desde su creación, por tanto, es *natural*, no añadido
- Todos los elementos en Javascript derivan de esa concepción
- Ofrece dos sintaxis para la definición de clases:
  - Basada en prototipos
  - Basada en la construcción `class`

Creación de clases basada en prototipos

```
<script>
function User(forename, username, password)
{
  this.forename = forename
  this.username = username
  this.password = password
  this.showUser = function()
  {
    document.write("Forename: " + this.forename + "<br />")
    document.write("Username: " + this.username + "<br />")
    document.write("Password: " + this.password + "<br />")
  }
}
</script>
```

Creación de objetos

```
userW = new User("Wolfgang", "w.a.mozart",
"composer")


userW = new User()
userW.forename = "Wolfgang"
userW.username = "w.a.mozart"
userW.password = "composer"
```

Con sintaxis de “class”

```
class User {
  constructor(forename, username, password ) {
    this.forename = forename;
  }

showUser ()
 {
   document.write("Forename: " +
      this.forename + "<br />")
   document.write("Username: " +
      this.username + "<br />")
   document.write("Password: " +
      this.password + "<br />")
 }
}
```

Palabra clave “prototype”

- Permite ahorrar espacio de memoria
  - `this.showUser = function()`
  - `User.prototype.showUser = function()`
- También para datos:
  - `User.prototype.greeting = “Hello”`
- Permite ampliar la funcionalidad de objetos predefinidos:
  - `String.prototype.trim = function()` …

Gestión de errores: excepciones

- Mecanismo básico de gestión de errores: excepciones
- Sintaxis y semántica similar a Java:
  - `try/catch/finally`

Ejemplo

```
function lastElement(array) {
  if (array.length > 0)
    return array[array.length - 1];
  else
    throw ”No puedo coger el último elemento de un” .
          “array vacío";
}

function lastElementPlusTen(array) {
  return lastElement(array) + 10;
}

try {
  print(lastElementPlusTen([]));
}
catch (error) {
  print(”Algo ha ido mal: ", error);
}
```

EJEMPLOS DE CÓDIGO

Temporizador

```
<!DOCTYPE html>
<html>
<body>

<p>Pulsa el botón para esperar 3 segundos; después sale un mensaje</p>
<button onclick="myFunction()">Pruébalo</button>

<script>
function myFunction()
{
setTimeout(function(){alert(”Hola")},3000);
}
</script>

</body>
</html>
```

Javascript para navegadores que no lo soportan

```
<SCRIPT>
    código javascript
</SCRIPT>
<NOSCRIPT>
Este navegador no comprende los scripts que se están ejecutando, debes
actualizar tu versión de navegador a una más reciente.
<br><br>
</NOSCRIPT>
```

Manejo de cookies

```
<!DOCTYPE html>
<html>
<head>
<script>
function getCookie(c_name)

function setCookie(c_name,value,exdays)

function checkCookie()

</script>
</head>
<body onload="checkCookie()">
</body>
</html>
```

getCookie

```
function getCookie(c_name)
{
var c_value = document.cookie;
var c_start = c_value.indexOf(" " + c_name + "=");
if (c_start == -1)
  {
  c_start = c_value.indexOf(c_name + "=");
  }
if (c_start == -1)
  {
  c_value = null;
  }
else
  {
  c_start = c_value.indexOf("=", c_start) + 1;
  var c_end = c_value.indexOf(";", c_start);
  if (c_end == -1)
    {
    c_end = c_value.length;
    }
  c_value = unescape(c_value.substring(c_start,c_end));
  }
return c_value;
}
```

setCookie y checkCookie

```
function setCookie(c_name,value,exdays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate() + exdays);
var c_value=escape(value) + ((exdays==null) ? "" : "; expires="+exdate.toUTCString());
document.cookie=c_name + "=" + c_value;
}

function checkCookie()
{
var username=getCookie("username");
if (username!=null && username!="")
  {
  alert("Welcome again " + username);
  }
else
  {
  username=prompt("Please enter your name:","");
  if (username!=null && username!="")
    {
    setCookie("username",username,365);
    }
  }
}
```

Principales bibliotecas en JS

(javascriptlibraries.com)

- Prototype: extensión de funcionalidad PDO
- script.aculo.us: Extensión para interfaces de usuario
- MooTools
- jQuery: document manipulation, navegation and animation
- DOJO
- DS3

Intercambio ligero de datos: JSON

- JavaScript Object Notation: formato de intercambio de datos que no requiere XML
- Es más ligero, más simple
- Más sencillo de usar y manejar
- Se puede usar conjuntamente con XML

Ejemplo de JSON: menú

```
{"menu": {
    "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
        {"value": "New", "onclick”:"CreateNewDoc()"},
        {"value": "Open", "onclick": "OpenDoc()"},
        {"value": "Close", "onclick": "CloseDoc()"}
      ]
    }
 }
}
```

Rizando el rizo … node.js

- Node.js
- Entorno de programación JavaScript para desarrollar aplicaciones que se ejecutan en el servidor

WebAssembly

- Lenguaje de programación en el cliente (servidor) de bajo nivel.
- Soporte para C/C++
- Formato objeto: árbol sintáctico abstracto, más rápido de analizar que JavaScript.

- https://webassembly.github.io

AJAX

Bibliografía

- D. Crane, E. Pascarello, D. James, «AJAX in Action», Manning, 2006
- A. Harris, «JavaScript & AJAX for Dummies», Wiley, 2010
- S. Jacobs, «Beginning XML with DOM and AJAX», Wrox Professional, Apress, 2006

Contenido

- Concepto
- Tecnologías componentes
- Principios
- Ejemplos de aplicaciones

AJAX

- **A**synchronous **J**avaScript **a**nd **X**ML

- Técnica de desarrollo web para crear **aplicaciones interactivas** de modo particular y gestionar **comunicación asíncrona** con el servidor
- Multiplataforma (S.O., navegador).
- Integra diversas tecnologías de desarrollo web usándolas de formas creativas y nuevas

Asíncrono

- En el contexto web significa que se pueden hacer simultáneamente. Por ejemplo, múltiples peticiones al servidor, independientes entre sí
- AJAX no implica obligatoriedad en la asincronía, pero suele ser así

XML (Datos)

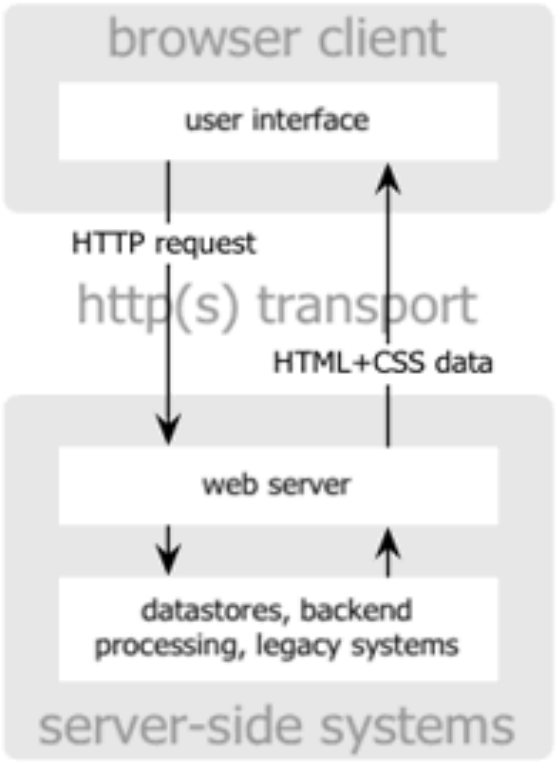
- El formato de datos habitual de intercambio es XML, pero no es exclusivo.
- Otras opciones (Tema 5):
  - Ficheros de texto
  - HTML formateado
  - JSON

¿Por qué surge AJAX?

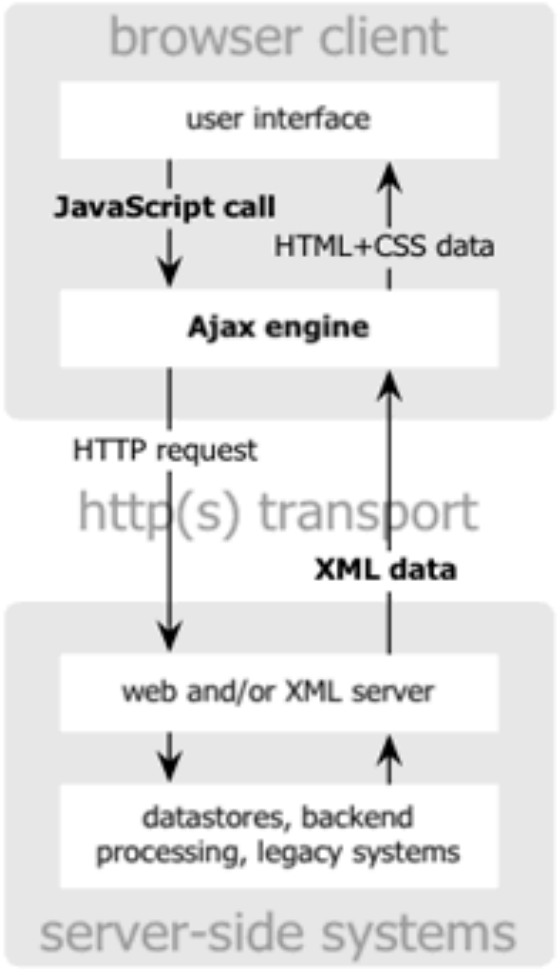
- Surge de la necesidad de empujar el desarrollo web a niveles mucho más lejanos de los disponibles
- Forzando el alcance de la tecnología mucho más allá de lo previsto en su creación
- Desarrollar aplicaciones web con la interactividad de las de escritorio
- Aplicación ilustrativa: carrito de la compra
  - Versión clásica: se envía una página para cada consulta
  - Versión AJAX: se envía sólo la información precisa, sin cambiar *toda* la página

AJAX en acción

- La comunicación con el servidor se realiza en el trasfondo, obteniendo sólo los datos realmente necesarios, en lugar de páginas completas
- La información se muestra instantáneamente sin refrescos de página, ni esperas
- Probar www.google.com

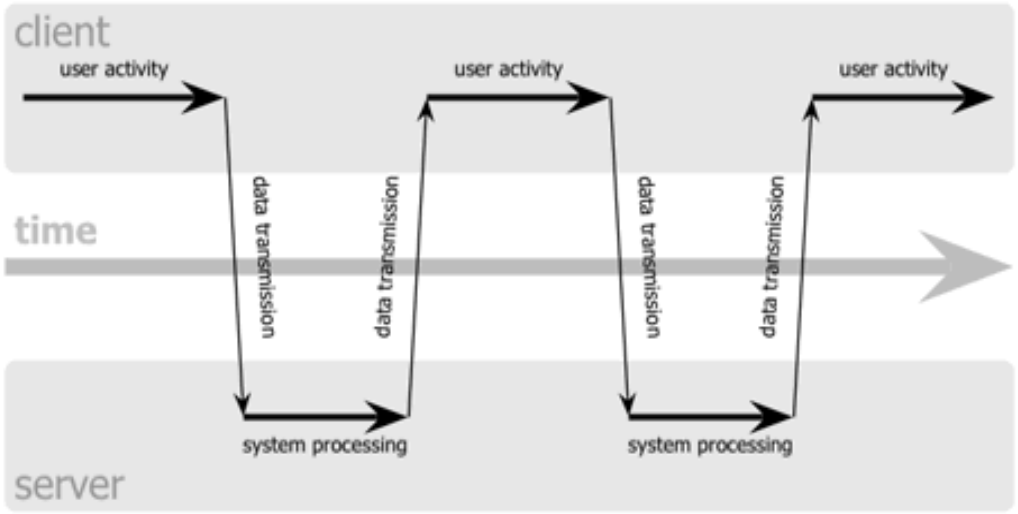


classic
web application model

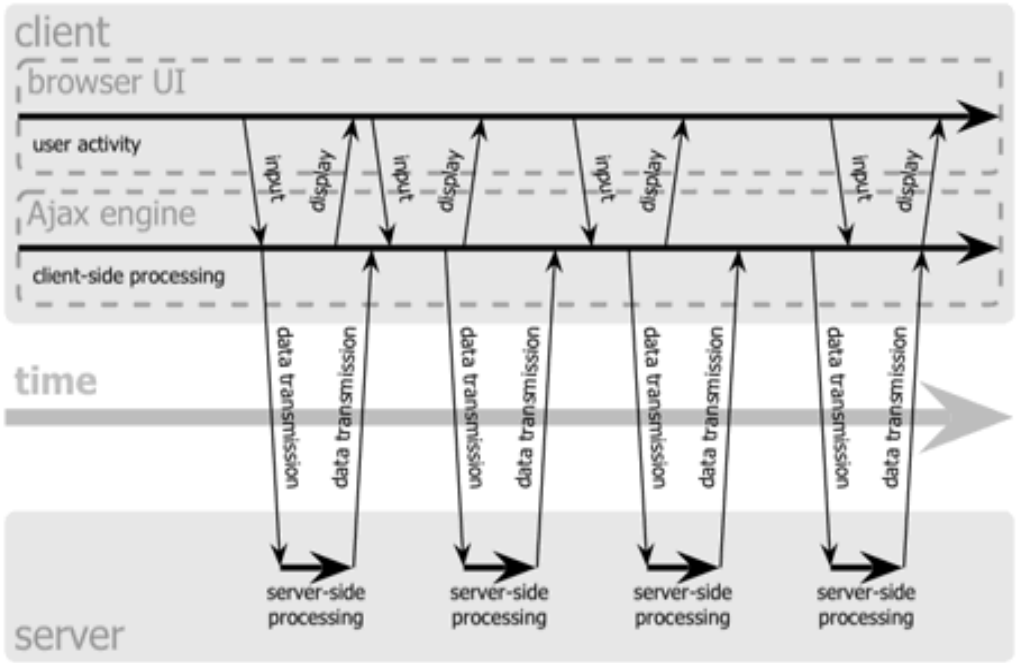


Ajax
web application model

classic web application model (synchronous)



Ajax web application model (asynchronous)



Tecnologías incluidas

- JavaScript
- DOM
- CSS
- XMLHttpRequest

XMLHttpRequest

- Objeto que no pertenece al estándar, pero soportan la mayoría de (implementaciones de JavaScript en) los navegadores
- Es un objeto para implementar una forma efectiva de hacer peticiones a un servidor web sin tener que recargar una página
- El objeto se controla con los métodos de la clase

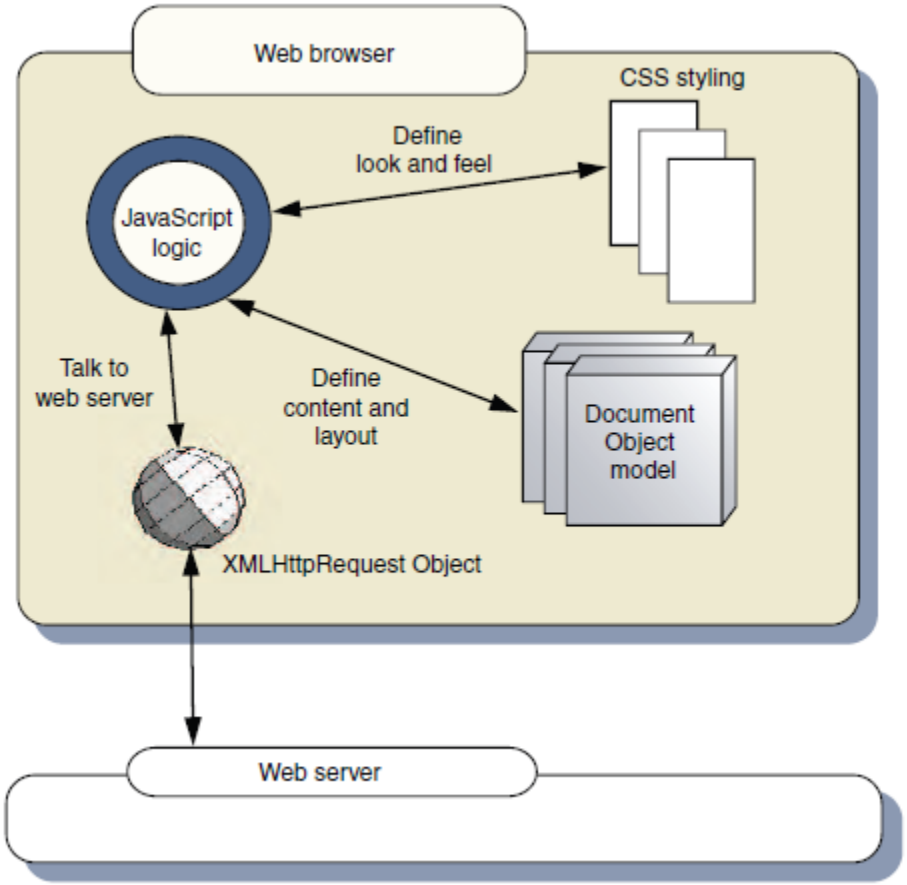
```
<!DOCTYPE html PUBLIC
“-//W3C//DTD XHTML 1.0 Strict//EN”
“http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd”>
<html lang = “EN” xml:lang = “EN” dir = “ltr”>
<head>
<meta http-equiv=“content-type” content=“text/xml; charset=utf-8” />
<title>asynch.html</title>
<script type = “text/javascript”>
//<![CDATA[
var request; //make request a global variable
function getAJAX(){
  request = new XMLHttpRequest();
  request.open(«GET», «beast.txt»);
  request.onreadystatechange = checkData;
  request.send(null);
}
function checkData(){
  if (request.readyState == 4) {
    // if state is finished
    if (request.status == 200) {
      // and if attempt was successful
     alert(request.responseText);
    }
  }
}
```

Ejemplo: Validacion de usuario (1)

```
<form>
  <p>
    Username: <input type="text”
      id="txtUserName" size="20”
      onblur="doCheck(this.value);"/>
    <span id="invalidMessage" class="invalid">
       </span>
  </p>
  <p>
    Password: <input type="text”
      id="txtPassword" size="20"/>
  </p>
</form>
```

Validación de usuario (2)

```
function doCheck(username) {
  if (username.length > 0) {
    document.getElementById("invalidMessage").innerHTML = "";
    if (window.XMLHttpRequest){
      xmlhttp=new XMLHttpRequest();
    }
    else if (window.ActiveXObject){
      try {
        xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
      } catch(e) {
      try {
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      } catch(e) {
      xmlhttp = false;
      }
    }
  }
  if (xmlhttp){
    xmlhttp.onreadystatechange=checkNames;
    xmlhttp.open("GET", "usernames.xml", true);
    xmlhttp.send(null);
    }
   }
}
```


Web browser
CSS styling
Define
look and feel
JavaScript
logic
Talk to
web server
Define
content and
layout
Document
Object
model
XMLHttpRequest Object
Web server

Clientes en JavaScript

- Un cliente en JavaScript combina datos, presentación y lógica de programa
- El aspecto se define con CSS
- La estructura de un documento es consultada y modificada mediante programación a través del DOM
  - La variable «document» es un enlace al nodo raíz
- Acceso asíncrono a los datos mediante XML

Principios definitorios de AJAX

1. El navegador alberga aplicaciones
2. El servidor provee datos
3. Interacción con el usuario fluida y continua
4. Aplicaciones AJAX

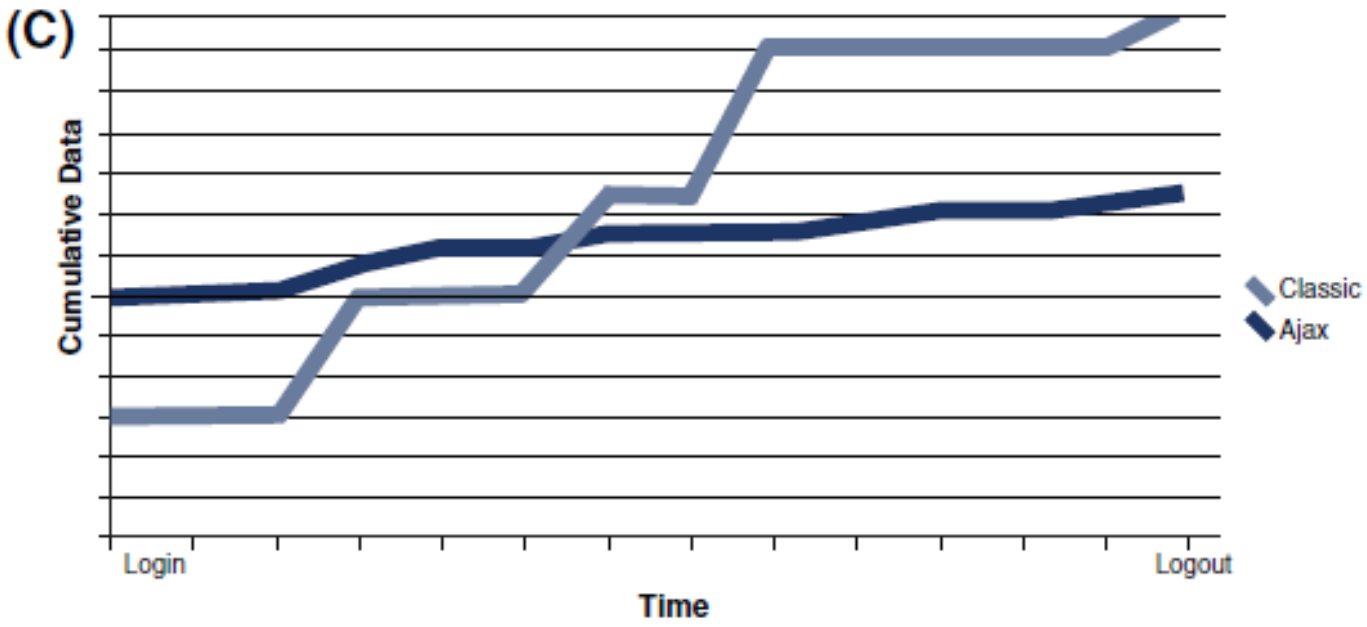
1. El navegador alberga aplicaciones

- En el modelo clásico de aplicación web, el navegador es como un terminal «tonto»
- En AJAX, parte de la lógica de la aplicación se traslada al navegador. Los documentos son más complejos
- El documento es persistente durante toda la sesión. Puede almacenar un estado

2. El servidor provee datos

- La interacción del usuario requiere del envío de pequeñas cantidades de datos
- El tráfico en aplicaciones AJAX tiene una carga fuerte inicial y después es muy reducida

Tráfico de red: AJAX vs clásica



3. Interacción del usuario fluida y continua

- Interacción en AWC: formularios e hiperenlaces
- «limbo» de interacción mientras se actualiza la página
- AJAX: interacción más fluida y continua: conectar funciones a eventos
- Conceptos de «drag-and-drop» acercan la experiencia web a la de aplicaciones de escritorio
- La comunicación no necesita una confirmación explícita

4. Aplicaciones AJAX

- Aplicación AJAX: aplicación informática en sentido estricto, no sólo pequeños scripts.
- Aplicación: modelado de datos, interacción con el usuario, procesamiento, comunicación con el servidor, generación de salida
- Aplicar buenas prácticas de desarrollo y programación: programación basada en patrones

Ejemplos de aplicaciones AJAX

- Google Apps:
  - Gmail
  - Google Maps
  - Google Suggest
  - www.flickr.com

Aplicaciones web clásicas vs. AJAX

- Aplicación web clásica: El flujo de trabajo se define por código en el servidor; el usuario va de una página a otra, con la recarga de páginas completas
- AJAX: El flujo de trabajo se define mediante software en el cliente y la comunicación el servidor se hace en el trasfondo mientras el usuario interacciona con el cliente
- La diferencia que imprime AJAX no es la tecnología sino el modelo de interacción que imprime a través del uso de las tecnologías

Browser Object Model (BOM)

- Incluido en las versiones 3.0 de los principales navegadores:
- permite acceder y modificar las propiedades de las ventanas del propio navegador
- es posible redimensionar y mover la ventana del navegador, modificar el texto que se muestra en la barra de estado y realizar muchas otras manipulaciones no relacionadas con el contenido de la página HTML
- Poca estandarización

Inconvenientes de AJAX

- Sobrecarga de la red (descargas iniciales; mayor número de conexiones)
- Dificultad para identificar cambios
- Las «nuevas» páginas no se registran en el historial de navegación
- El contenido generado no es indizado por motores de búsqueda
- No es totalmente portable

Bibliotecas AJAX

- DOJO: widgets para la GUI
- MochiKit: Facilita la programación en JavaScript, permitiendo un estilo similar a Python
- Prototype: Soporte para AJAX y sus extensiones
- jQuery: Facilita el desarrollo de código en JavaScript

Alternativas a AJAX

- Flash
- Java Web Start

  (p.ej.: SAETA)