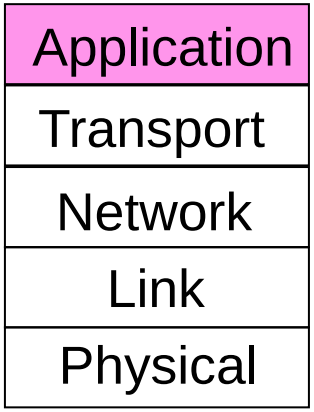
Capítulo 2

La Capa de Aplicación: Parte 3



La Web: Metas

- **Vamos a estudiar**:
  - **Críticas al modelo de páginas dinámicas**
  - Procesamiento de eventos con JavaScript
  - Modificación de páginas con JavaScript
  - Manejo de pedidos y respuestas HTTP

Críticas al modelo actual

- **Que el servidor web tenga que construir páginas dinámicas puede ser ineficiente por los siguientes motivos:**
  1. La página nueva a generar dinámicamente en el servidor puede tener una parte importante en común con la página que ya tiene el browser;
     - Y esa parte que se repite se genera de nuevo y tiene que enviarse de nuevo por la red.
     - Esa parte repetida va a tener que ser interpretada de nuevo por el browser.

Críticas al modelo actual

2. El cliente se queda bloqueado esperando luego de hacer un pedido HTTP al servidor web y recién puede continuar ejecutándose cuando recibe una página (estática o generada dinámicamente).
   - Estos son los llamados **pedidos sincrónicos**.
   - Si el procesamiento de un pedido del lado del servidor toma mucho tiempo, el no poder usar la aplicación web mientras tanto para otra cosa puede ser bastante desagradable para el usuario.

Críticas al modelo actual

- **Problema**: ¿Cómo evitar tener que, regenerar, reenviar y recargar partes repetidas de páginas, y evitar que el cliente se quede bloqueado demasiado tiempo esperando por la respuesta a un pedido HTTP?

Liberando el servidor web y aprovechando más el poder del cliente

- **Solución**: El servidor web en respuesta a un pedido HTTP solo enviará datos (**no genera** página web con esos datos) para actualizar parte de una página web en el navegador (sin tener que recargar una página completa) y el browser se ocupe de actualizar la interfaz del usuario (IU) con esos datos. Que los pedidos sean asincrónicos.
  - Cuando llegan datos del servidor web se genera e inserta la parte nueva de la IU y se deja igual la parte de la IU que debe preservarse.
  - **Pedidos asincrónicos**: El cliente hace un pedido al servidor web y el cliente puede seguirse ejecutando para otra cosa, mientras el pedido es procesado del lado del servidor.

Liberando el servidor web y aprovechando más el poder del cliente

- **Implementación de la solución**:
  - Primero se pide el código de la IU de la aplicación al servidor Web
    - dicho código está escrito usando HTML + JavaScript + pedidos HTTP asincrónicos.
    - Esa IU tiene una apariencia similar a las IU de aplicaciones de escritorio (a esto se le llama **IU enriquecida**).
  - Cada vez que se hace un pedido asincrónico, se reciben datos del servidor.
    - Esos datos son procesados por el cliente para actualizar la IU.
- Por lo tanto estudiaremos:
  - JavaScript para actualizar la IU,
  - cómo hacer pedidos HTTP (asincrónicos) y
  - cómo procesar respuestas HTTP del lado del cliente.

La Web: Metas

- **Vamos a estudiar**:
  - Críticas al modelo de páginas dinámicas
  - **Procesamiento de eventos con JavaScript**
  - Modificación de páginas con JavaScript
  - Implementación de pedidos y respuestas HTTP

JavaScript

Un poquito de sintaxis fundamental de JavaScript

Construcción	**Description**
<script> … </script>	Para insertar un script en JavaScript
function NAME (PARAMETERS) {BODY}	Declaración de funciones
Var NAME = EXPR;	Declaración e inicialización de variables
Objeto document	Representa la página siendo visualizada
Document.write(t)	Escribe texto *t* en el stream de salida de la página siendo visualizada.
Método document.getElementById(*X*)	Retorna elemento de identificador *X.* *Es el elemento con atributo id y valor X.*
Propiedad *e*.innnerHTML	Contenido del elemento *e*

JavaScript

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

<h2>My First Page</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello
World!";
</script>

</body>
</html>
```

Aclaraciones:

- Elementos HTML pueden tener **atributos id.**
- El valor de un atributo id debe ser único en todo el documento HTML y sirve para referirse a un elemento unívocamente.
- Se cambia el contenido en el párrafo de con id=“demo”.
- Esta es la forma típica de reemplazar el contenido de un elemento por otro.

Procesamiento de eventos en JavaScript

- **Problema**: ¿cómo procesar eventos producidos por el usuario sobre elementos como áncoras, botones, campos de formulario, etc.?
  - La reacción a un evento puede significar hacer cambios en la IU, o hacer cierta tarea de procesamiento en el cliente o hacer pedidos al servidor web y mostrar las respuestas en pantalla.
- **Solución**: Definir **reglas ECA** (reglas evento-condición-acciones): si ocurre un evento, y se da una condición, entonces ejecutar una secuencia de acciones.
- **¿Cómo implementar una regla ECA en JavaScript?**

Procesamiento de eventos en JavaScript

- **Nombres de eventos**: `abort`, `click`, `focus`, `load`, `select`, `submit`, `unload`, etc.
- Los eventos ***se asocian*** a etiquetas de la página HTML.
  - Para ello insertar en elemento atributo: on[nombre-evento]
  - <[nombre-elemento] on[nombre-evento]= "…">.
  - P. ej: <button onclick = "..." >
- La parte de acciones es el procesamiento de un evento en sí:
  - Cuando ocurre el evento se trata de ejecutar una secuencia de comandos JavaScript o ejecutar una función de JavaScript.
  - Poner:
    - <[nombre-elemento] on[nombre-evento] = "[acciones]" >
    - <[nombre-elemento] on[nombre-evento] = "[nombre-función()]">
  - P.ej: <button onclick="MostrarMailsEnviados()">
  - Funciones JavaScript: para procesar eventos se pueden encerrar en etiquetas <script> colocadas dentro de etiqueta <head>.

Procesamiento de Eventos con JavaScript

- Ejemplo de procesamiento de evento con una función:

```
<html>
 <head>
     <title> Ejemplo de manejo de eventos </title>
      <script>
          function manejador(){
          document.write('mensaje del manejador') }
       </script>
  </head>
  <body id="cuerpo" onload="manejador()">
          <p> mensaje normal </p>
  </body>
</html>
```

Procesamiento de Eventos con JavaScript

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

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

Click on this text!

Después de hacer click en el texto:

Ooops!

Para insertar una regla ECA ir al elemento HTML correspondiente y escribir on<event> =“*f*(x)”, donde <event> es nombre de evento y *f* es una función definida en JavaScript.

La Web: Metas

- **Vamos a estudiar**:
  - Críticas al modelo de páginas dinámicas
  - Procesamiento de eventos con JavaScript
  - **Modificación de páginas con JavaScript**
  - Implementación de pedidos y respuestas HTTP

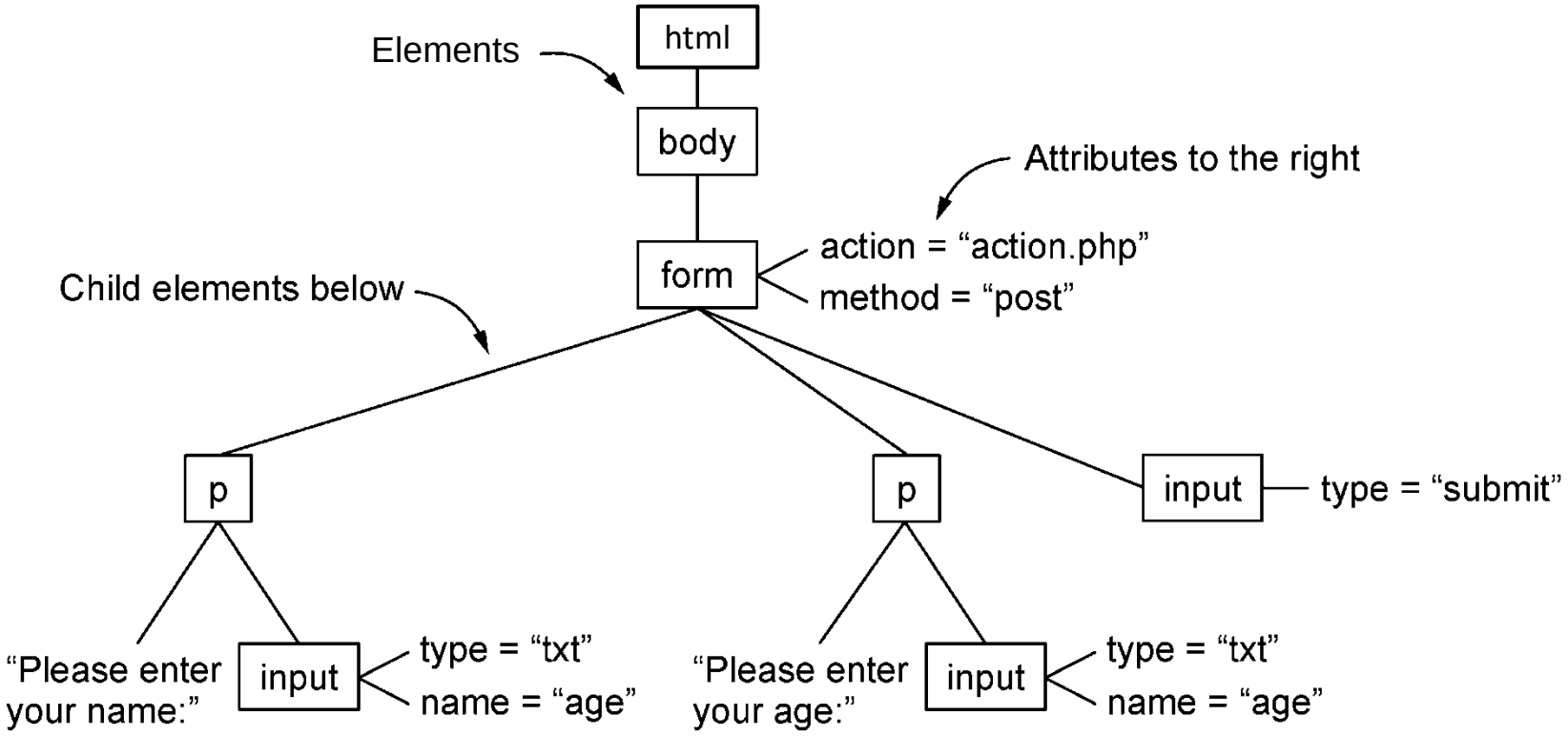
DOM

- **DOM** es una representación de una página HTML que es accesible a programas.
- Esta representación es estructurada como un **árbol DOM** que refleja la estructura (jerárquica) de los elementos HTML.
  - ❑ En la raíz está el elemento *HTML* que representa la página.
  - ❑ El elemento *HTML* es el padre de elementos *head* y *body*.
  - ❑ El elemento *body* a su vez puede ser padre de elementos *form*, *ol*, *ul*, etc.
  - ❑ Además al lado de un elemento puede uno dibujar sus **atributos**. (p.ej. para *form* los atributos *action* y *method*).

DOM

```
<html>
<body>
  <form action="action.php" method="post">
    <p> Please enter your name: <input type="text" name="name"> </p>
    <p> Please enter your age: <input type="text" name="age"> </p>
        <input type="submit">
  </form>
</body>
</html>
```

DOM



El árbol DOM del HTML anterior

Modificación de una página usando JavaScript

- **Importancia del modelo DOM**: permite a los programas **cambiar** partes de la página.
  - No hay necesidad de reescribir la página entera.
- JavaScript puede acceder al modelo DOM de una página.
- Además se pueden tener scripts que modifican el documento HTML siendo visualizado.

Modificación de una página usando JavaScript

- JavaScript usa ‘.’ para **navegar** a través de las propiedades y referencias asociadas con el documento.
  - La **jerarquía de objetos** del árbol DOM es navegada comenzando con **objeto window** que representa al navegador.
  - Este objeto tiene una **propiedad** llamada **document** que representa la página HTML.
  - El **objeto document** tiene una colección de formularios llamada **forms** indexada por el número de aparición dentro del documento comenzando desde 0.
  - Un **formulario** tienen una colección de elementos (de entrada) llamada **elements** indexada por el número de aparición dentro el formulario comenzando desde 0.
- Ejemplo:
  - window.document.forms[0].elements[3].value = “lunes”;
  - Fija valor de elemento 4 (correspondiente al día) del primer formulario en “lunes”.

Contenedores

- La etiqueta ** ** define una división o sección en un documento HTML.
- El elemento es usado a menudo como un contenedor para otros elementos HTML.
- Elementos pueden anidarse unos dentro de otros lo que permite definir jerarquía de contenedores y organizar una página.
- Ejemplo:

```
<div style="background-color:lightblue">
  <h3>This is a heading</h3>
  <p>This is a paragraph.</p>
</div>
```

Modificación de una página usando JavaScript

- Algunas sentencias y métodos que permiten modificar una página HTML siendo visualizada.

Construcción	**Description**
e.innerHTML = c;	Cambia contenido HTML de *e* por *c*
e.a = v;	Cambia valor de atributo *a* de elemento *e* por *v*.
e.setAttribute(a,v)	Crea en elemento *e* el atributo *a* con valor *v*. Si *a* ya estaba , solo cambia su valor.
document.createElement(*N*)	Crea elemento HTML de etiqueta *N*
document.createTextNode(*T*)	Crea nodo de texto *T*
e.appendChild(n)	Se agrega elemento *n* como último hijo del elemento *e*
e.removeChild(n)	De elemento *e* se remueve subelemento *n.*
e.replaceChild(n1,n2)	De elemento *e* se reemplaza subelemento *n1* por elemento *n2*.

Modificación de una página usando JavaScript

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

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>

</body>
</html>
```

This is a paragraph.

This is another paragraph.

This is new.

Aclaraciones: el script crea el elemento <p>, luego crea un nodo de texto, luego agrega el nodo de texto al elemento <p> creado. Finalmente el elemento <p> creado es agregado como último hijo del contenedor de identificador “div1”.

Modificación de una página usando JavaScript

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

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
parent.removeChild(child);
</script>

</body>
</html>
```

This is another paragraph.

Aclaración: Para remover de elemento identificado por “div1” el párrafo identificado por “p1”: obtener el padre de “p1” que es “div1”, obtener el elemento “p1”; finalmente se remueve del padre de “p1” el elemento “p1”.

Modificación de una página usando JavaScript

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

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>

</body>
</html>
```

This is new.

This is another paragraph.

Crea un elemento <p> de contenido “this is new” y lo reemplaza por elemento de nombre “p1”

La Web: Metas

- **Vamos a estudiar**:
  - Críticas al modelo de páginas dinámicas
  - Procesamiento de eventos con JavaScript
  - Modificación de páginas con JavaScript
  - **Implementación de pedidos y respuestas HTTP**

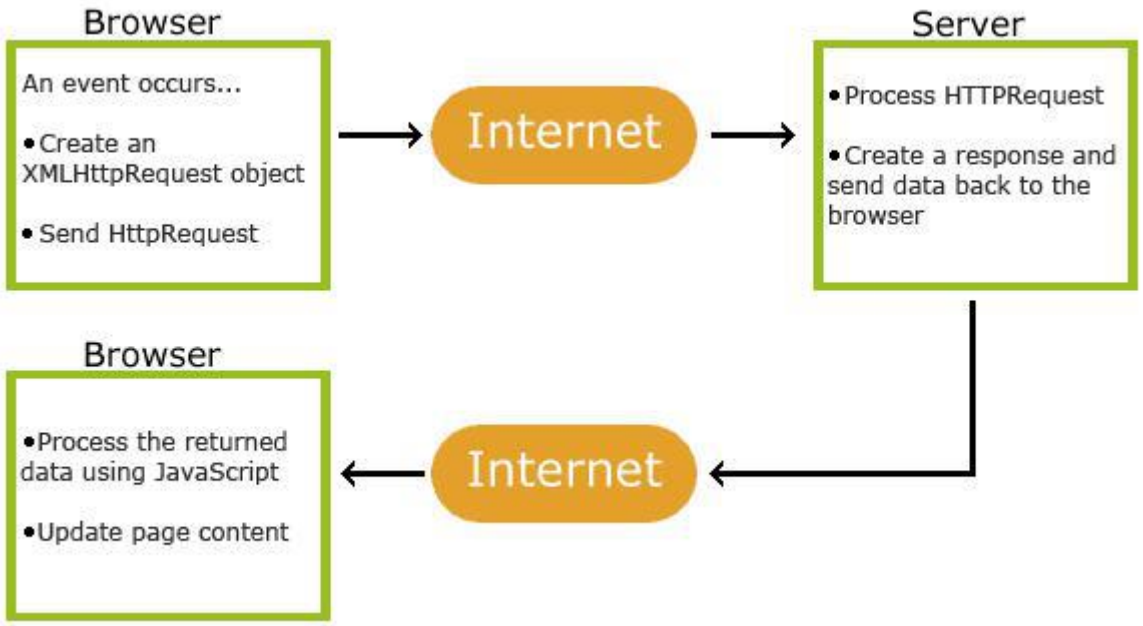
Manejo de pedidos y respuestas HTTP

Primitivas para hacer y procesar pedidos/respuestas HTTP

Construcción	Description
Objeto XMLHttpRequest	Objeto usado para intercambiar datos con servidor web.
Método Open(method, url, async, user, psw)	Para especificar el pedido HTTP
Método send()	Para enviar el pedido HTTP
Método setRequestHeader()	Para fijar encabezados del pedido HTTP
Propiedad responseText de objeto XMLHttpRequest	Se refiere a la respuesta del servidor como un string
Propiedad readyState	Estatus del XMLHttpRequest
getAllResponseHeaders()	Retorna toda la información de encabezados de respuesta

Manejo de pedidos/respuestas HTTP

Pasos para hacer pedido HTTP y contestarlo:



1. Se especifica pedido HTTP usando *open*().
2. Un pedido HTTP se envía con *send*() y va a ser un pedido de datos o texto.
3. El servidor va a obtener los datos pedidos (p.ej. de una base de datos), con ellos va a crear una respuesta HTTP y enviarla al navegador.
4. El cliente recibe la respuesta al pedido, procesa los datos recibidos y modifica la página actual (usando las primitivas para modificar la IU que acabamos de ver.)

Manejo de pedidos HTTP

- **Manejo de pedidos http al servidor:**
  - El **objeto XMLHttpRequest** puede usarse para intercambiar datos con un servidor web.
  - La sintaxis para **crear un objeto XMLHttpRequest**:
    - *variable* = new XMLHttpRequest();
    - P.ej: var xhttp = new XMLHttpRequest();
  - Para especificar el **pedido http** usar el método:
    - open(*method, url, async, user, psw*)
    - Method es GET o POST, url es localización del archivo, async es true si pedido asincrónico y false sino, luego user name y password.
    - **Pedidos asincrónicos**: el cliente hace un pedido y sigue trabajando.
    - **Pedidos sincrónicos**: el cliente hace un pedido y se bloquea esperando por una respuesta.

Manejo de pedidos HTTP

- **Manejo de pedidos http al servidor:**
  - Para **enviar pedido GET** al servidor usar método send().
  - P. ej:

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);
xhttp.send()
```

  - Para **enviar pedido POST** al servidor usar send(string).
  - Para **fijar encabezados** usar método: setRequestHeader()
  - P.ej: para postear datos como un formulario HTML:

```
xhttp.open("POST", "ajax_test.asp", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

Manejo de respuestas HTTP

- **Manejo de respuestas HTTP del servidor:**
  - la **propiedad responseText** retorna la respuesta del servidor como un string.
  - P.ej:
    - `document.getElementById("demo").innerHTML = xhttp.responseText;`
    - **document.getElementById()** retorna el elemento de id especificado.
    - Para modificar el contenido de un elemento HTML se usa la propiedad **innerHTML**.

Manejo de respuestas HTTP

- **Manejo de respuestas HTTP del servidor:**
  - La propiedad **readyState** mantiene el **estatus del XMLHttpRequest.**
  - La **propiedad onreadystatechange** define una función a ser ejecutada cuando el **readyState** cambia.
  - **Cambios en estado** pueden ser por ejemplo:
    - 1: Conexión con el servidor establecida
    - 2: Pedido recibido
    - 3: Procesando pedido
    - 4: El pedido terminó y la respuesta está lista

Manejo de respuestas HTTP

- P.ej:

```
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML =
       this.responseText;
  }
};
```

- Si el pedido terminó, la respuesta está lista y el pedido fue exitoso se pone como contenido HTML del elemento de identificador “demo” el texto de this.responseText.

Manejo de respuestas HTTP

- **Manejo de respuestas HTTP del servidor:**
  - El método **getAllResponseHeaders()** retorna toda la **informacion de encabezados** de la respuesta del servidor.
    - `document.getElementById("demo").innerHTML = this.getAllResponseHeaders();`
  - El método **getResponseHeader()** retorna un **encabezado específico** de la respuesta del servidor.
    - `document.getElementById("demo").innerHTML = this.getResponseHeader("Last-Modified");`

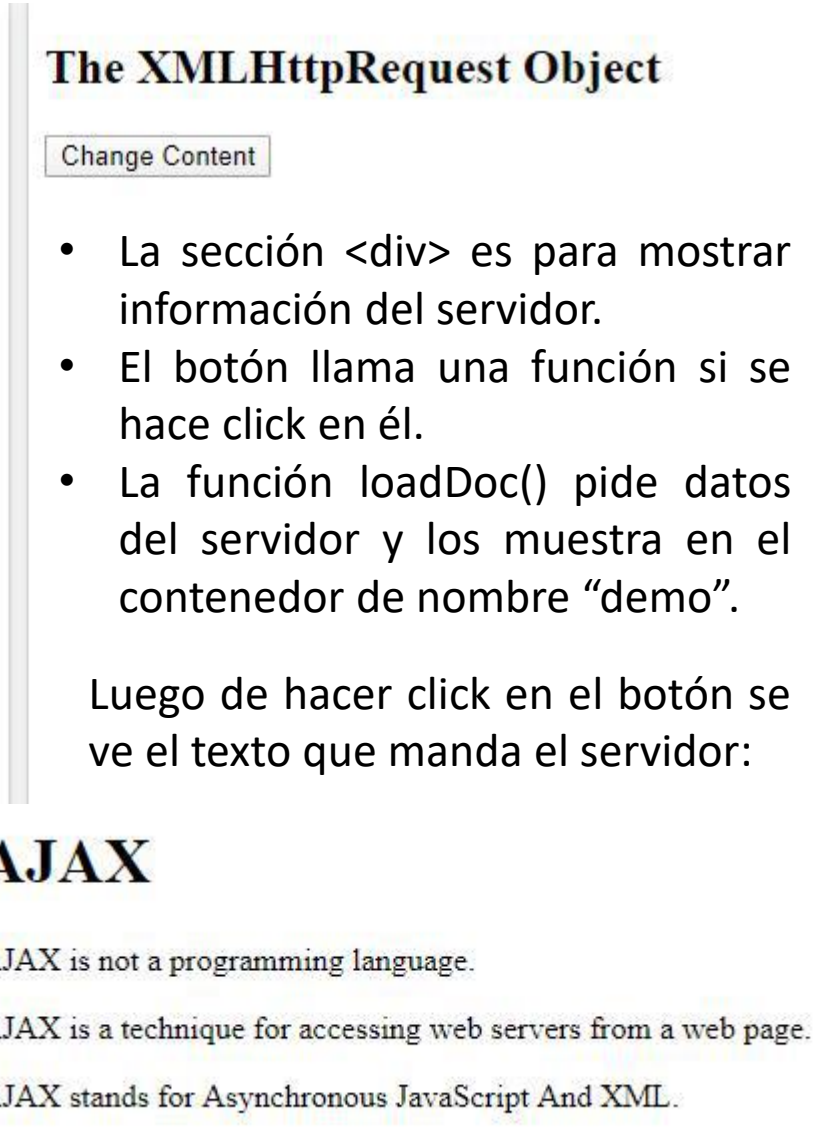
Manejo de pedidos/respuestas HTTP

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

<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change
Content</button>
</div>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>

</body>
</html>
```



- La sección es para mostrar información del servidor.
- El botón llama una función si se hace click en él.
- La función loadDoc() pide datos del servidor y los muestra en el contenedor de nombre “demo”.

Luego de hacer click en el botón se ve el texto que manda el servidor:

Manejo de pedidos/respuestas HTTP

```
<CATALOG>
 <CD>
   <TITLE>Empire Burlesque</TITLE>
   <ARTIST>Bob Dylan</ARTIST>
   <COUNTRY>USA</COUNTRY>
   <COMPANY>Columbia</COMPANY>
   <PRICE>10.90</PRICE>
   <YEAR>1985</YEAR>
 </CD>
 <CD>
   <TITLE>Hide your heart</TITLE>
   <ARTIST>Bonnie Tyler</ARTIST>
   <COUNTRY>UK</COUNTRY>
   <COMPANY>CBS Records</COMPANY>
   <PRICE>9.90</PRICE>
   <YEAR>1988</YEAR>
 </CD>
 <CD>
   <TITLE>Greatest Hits</TITLE>
   <ARTIST>Dolly Parton</ARTIST>
   <COUNTRY>USA</COUNTRY>
   <COMPANY>RCA</COMPANY>
   <PRICE>9.90</PRICE>
   <YEAR>1982</YEAR>
 </CD>
 ...
<CATALOG>
```

XML (Extensible Markup Language)

XML se puede usar para definir datos así:

- Los *nombres* de los elementos de datos son expresados mediante etiquetas del tipo: <nombre>.
- Un elemento de datos de nombre *n* consiste de información que se encierra entre etiquetas: <n> y </n>.
- Esa información puede construirse usando otros elementos de datos anidados entre <n> y </n> o solo texto.
- P.ej: en <TITLE>Greatest Hits</TITLE> el elemento <TITLE> tiene solo información de texto.
- P.ej: elementos de nombre <ARTIST> anidados en elementos de nombre <CD>.

Manejo de pedidos/respuestas HTTP

- La propiedad **responseXML** retorna la respuesta del servidor como un documento XML.
- La función **d.getElementsByTagName(*N*)** retorna los elementos XML dentro del documento *d*, que tienen nombre de etiqueta *N*.
- Se puede **recorrer un elemento XML**:
  - Al igual que cuando hablábamos del DOM, se puede pensar el documento XML como un árbol y se puede recorrer ese árbol usando **expresiones de camino** donde cada sección de una expresión de camino se separa con '.'.
  - **e.childNodes[i]** retorna el nodo hijo *i* del elemento *e.* (el *i*-ésimo subelemento dentro de *e*).
  - Si *e* es un elemento con contenido de solo texto, ***e*.nodeValue** retorna el texto del contenido de *e*.
  - Recordar que un documento HTML5 respeta XML. O sea, que nada impide generalizar la idea del DOM de HTML5 a XML.
  - P.ej: x[i].childNodes[0].nodeValue
    - es una expresión de camino donde primero se toma elemento i-esimo dentro de x, luego para ese elemento se toma primer subelemento anidado que es de solo texto y luego se toma ese texto.

Manejo de pedidos/respuestas HTTP

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

<h2>The XMLHttpRequest Object</h2>

<p id="demo"></p>

<script>
var xhttp, xmlDoc, txt, x, i;
xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    xmlDoc = this.responseXML;
    txt = "";
    x = xmlDoc.getElementsByTagName("ARTIST");
    for (i = 0; i < x.length; i++) {
      txt = txt + x[i].childNodes[0].nodeValue + "
<br>";
    }
    document.getElementById("demo").innerHTML = txt;
  }
};
xhttp.open("GET", "cd_catalog.xml", true);
xhttp.send();
</script>

</body>
</html>
```

The XMLHttpRequest Object

Bob Dylan
Bonnie Tyler
Dolly Parton
Gary Moore
Eros Ramazzotti
Bee Gees
Dr.Hook
Rod Stewart
Andrea Bocelli
Percy Sledge
Savage Rose
Many
Kenny Rogers
Will Smith
Van Morrison
Jorn Hoel
Cat Stevens
Sam Brown
T'Pau
Tina Turner
Kim Larsen
Luciano Pavarotti
Otis Redding
Simply Red
The Communards
Joe Cocker

Manejo de pedidos/respuestas HTTP

- **Aclaraciones sobre el ejemplo anterior:**
  - En variable *xmlDoc* se asigna la respuesta XML que es un documento XML.
  - En la variable *x* se asignan todos los elementos de etiqueta ARTIST.
  - En la variable *txt* se cargan todas los nombres de artistas presentes en *x*.
    - Para eso hay que recorrer *x* (lo que se logra con un for de JavaScript).
  - Esa lista de artistas es asignada al párrafo de id="demo".

Ajax

- Hemos estado usando el enfoque Ajax.
- **AJAX** (**Asynchronous JavaScript and XML**) es un conjunto de tecnologías que trabajan juntas para permitir que las aplicaciones web sean tan eficientes y poderosas como las aplicaciones desktop tradicionales. Estas tecnologías son:
  1. HTML y CSS para presentar información como páginas.
  2. **DOM** (**Domain Object Model**) para cambiar partes de las páginas mientras son vistas.
  3. **XML** (**Lenguaje extensible de marcas**) para permitir que los programas intercambien datos de aplicación con el servidor.
  4. **Comunicación asincrónica** del cliente con el servidor web para envíar y retornar datos (p.ej. en XML).
  5. **JavaScript** como lenguaje para ligar todas estas facilidades juntas.