http://www.flickr.com/photos/kosmar/62381076

Tema 3

Javascript parte II: AJAX

Asynchronous **J**avaScript **a**nd **X**ML. O cómo hacer peticiones al servidor desde Javascript

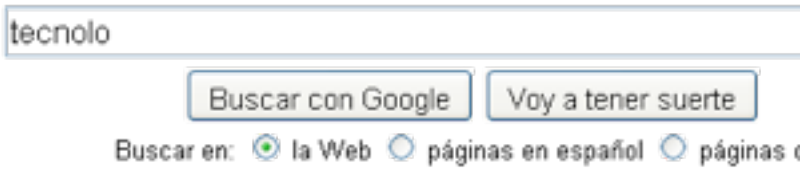
AJAX (Asynchronous JavaScript And XML)

- **Combinación de tecnologías** que permite:
  - Hacer peticiones al servidor con javascript y recibir la respuesta sin recargar la página ni cambiar de página
  - Para insertar en la página la información que envía el servidor se usa el API del DOM, que ya veremos

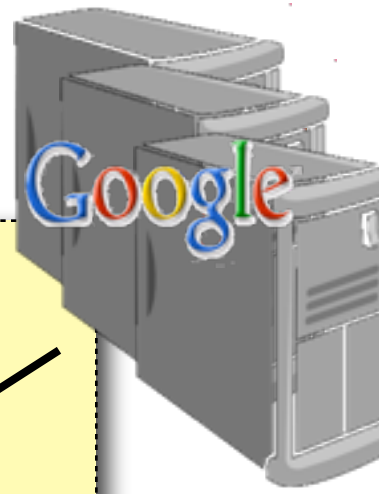
- El API existe desde hace algunos años tanto en IE como en Firefox, pero lo "pusieron de moda" los/as muchachos/as de Google (suggest, Gmail,…)
  - Actualmente es una de las características distintivas de las aplicaciones web "2.0"

Cómo podría funcionar Google *suggest*

- El código real es complicadillo…
- Esto es un ejemplo "inventado". No estamos usando el API real de AJAX




```
<input type="text" onkeyup="suggest(this.value)"/>
<script type="text/javascript">
 function suggest(texto) {
    hacerPeticionHTTP("www.google.es?sugerencia="+texto)
    resp =obtenerRespuestaHTTP();
    muestraResultados(resp)
 }
 function muestraResultados(resp) {
   //crear dinámicamente un div con un recuadro alrededor
   //en el que aparezca la información contenida en "resp"
 }
</script>
```



Petición HTTP

Respuesta HTTP

Tecnologia 1650000000
Tecnología led 1780000
…

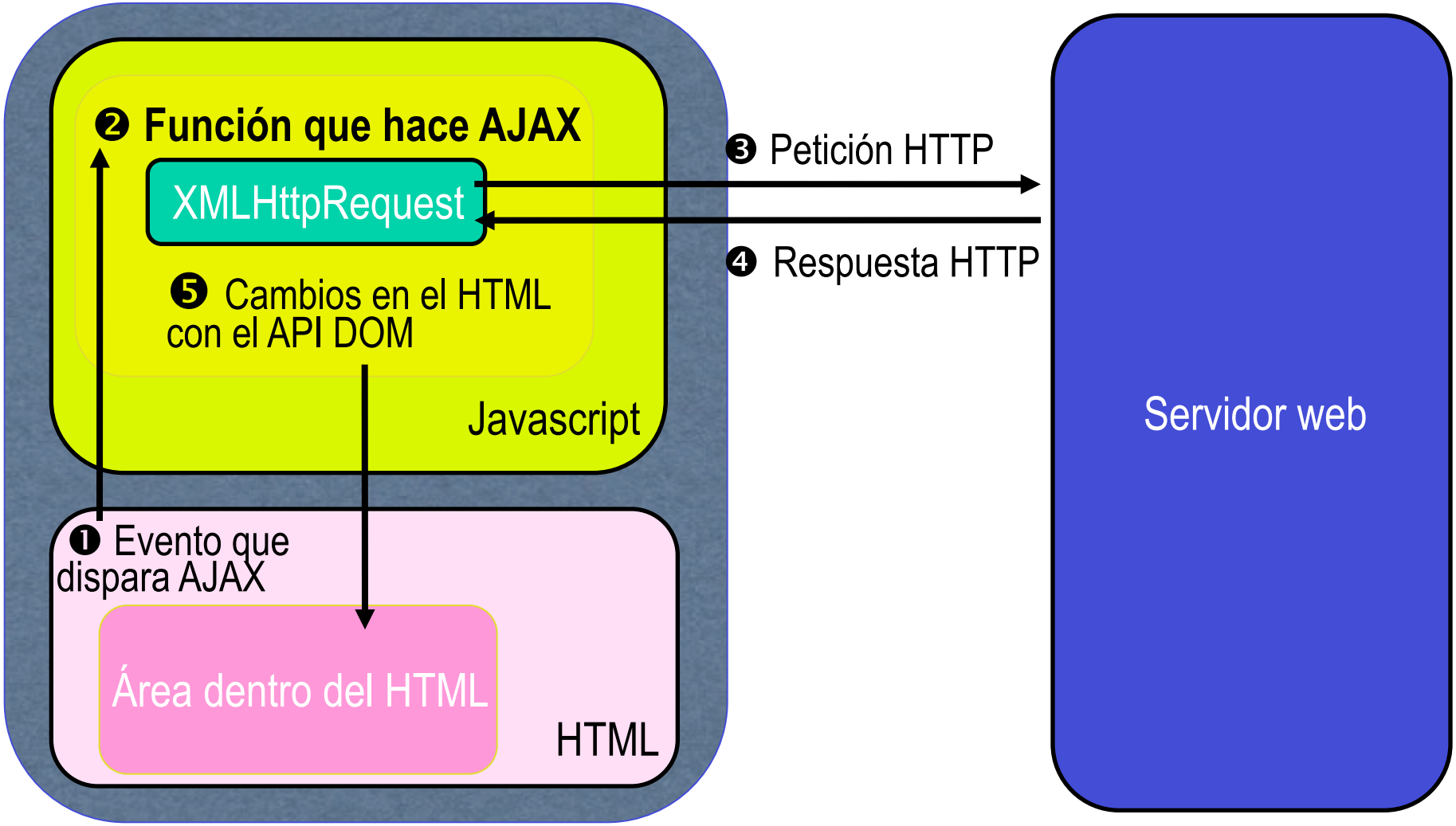
El API de AJAX

- Todo gira en torno a la "clase" XMLHttpRequest
  - Original de Explorer (!), aunque luego adoptada por el resto de navegadores. Actualmente en proceso de estandarización (W3C)
  - En Explorer 6, XMLHttprequest no es un objeto nativo

```
var xmlHttp = null;
if (window.XMLHttpRequest)
    // IE7,IE8, resto de navegadors
    xmlHttp = new XMLHttpRequest();
else    if (window.ActiveXObject)
            // ActiveX (IE6)
            xmlHttp = new ActiveXObject('MSXML2.XMLHTTP.3.0');
else alert("tu navegador no es compatible con AJAX");
```

- XMLHttpRequest tiene métodos para
  - Lanzar la petición HTTP
  - Comprobar el código de estado devuelto por el servidor

Esquema AJAX síncrono



Código síncrono

- **ACLARACION**: El AJAX normalmente es **asíncrono** (¿si no, qué significa la primera "A" del acrónimo?), aunque veremos primero este caso por ser más simple
- Paso 1: Crear el objeto XMLHttpRequest
- Paso 2: Realizar la petición
  - Método open: Prepararla (diferente según sea GET/POST y síncrona/asíncrona). En caso de ser GET los parámetros van aquí
  - Método send: Enviarla (en caso de ser POST los parámetros van aquí)
- Paso 3: Procesar la respuesta
  - La propiedad status del objeto XMLHttpRequest contiene el código de estado del servidor (en HTTP, OK es el 200)
  - La respuesta del servidor la tenemos en la propiedad responseText

Ejemplo de código síncrono

- Supongamos que en el servidor esperan un parámetro "cod" con un valor numérico que tenemos en la variable Javascript "codigo".

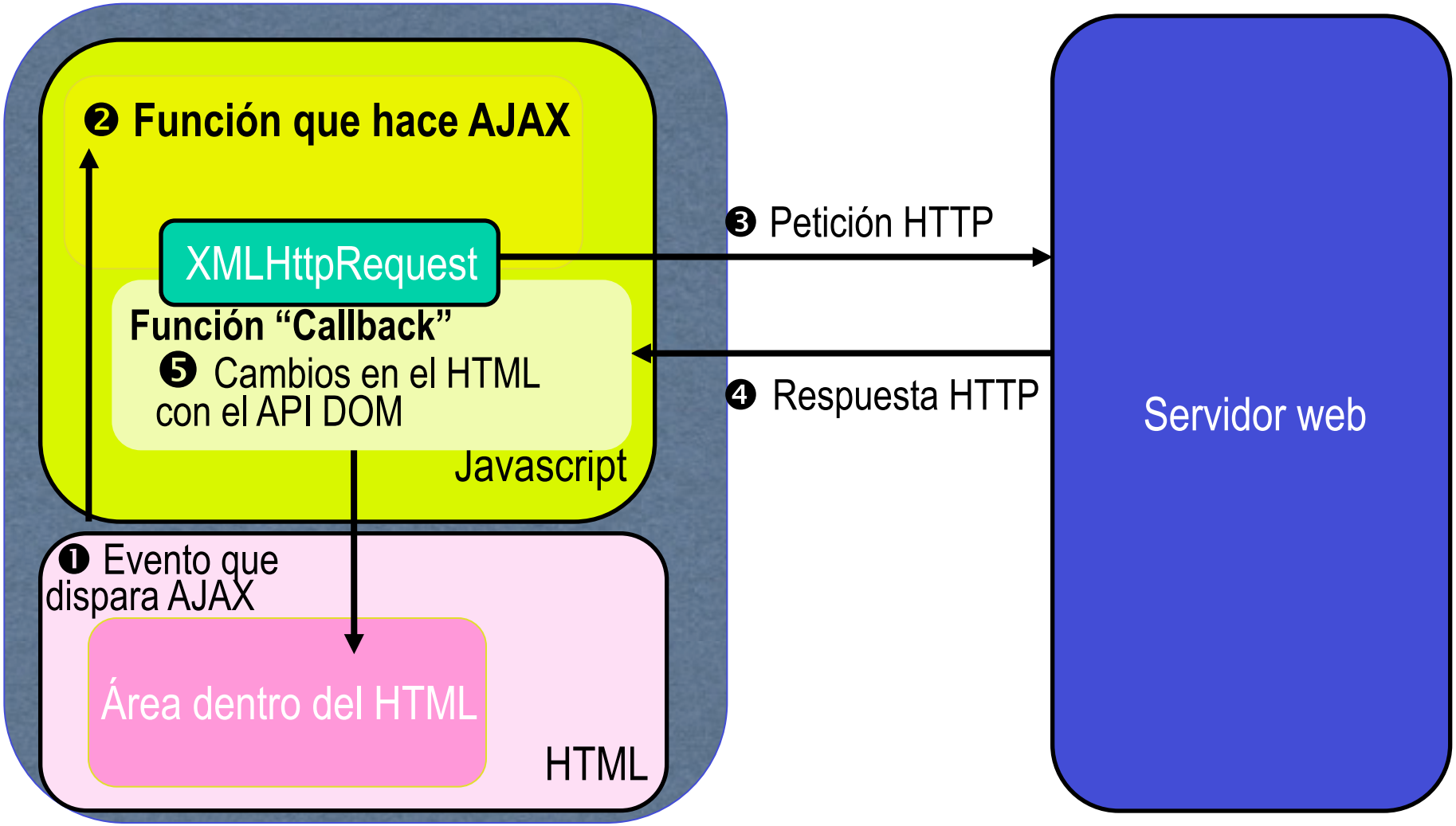
```
//crear el objeto "xmlhttprequest", versión simplificada que no iría en IE6
var req = new XMLHttpRequest();
//preparar la petición. El tercer parámetro indica que no es asíncrona
req.open('GET', 'http://www.miservidor.com/miprograma.php?cod=' +codigo, false);
//Enviar la petición. Con GET el argumento siempre es null
//esta instrucción se bloquea hasta que no llegue la respuesta del servidor (al ser síncrona)
req.send(null);
//comprobar el código de estado
if(req.status == 200)
//normalmente haremos algo más útil que un simple alert
alert(req.responseText)
```

versión con GET

```
//En la versión con POST sería todo igual excepto "open" y "send"
//con POST los argumentos no van en la URL, sino en el "send"
…
req.open('POST', 'www.miservidor.com/miprograma.php', false);
req.send('cod='+codigo);
```

versión con POST

Esquema AJAX **asíncrono**



Código asíncrono

- El Javascript puede seguir haciendo "otras cosas" mientras se recibe la respuesta del servidor.
  - En el caso síncrono el usuario podría pensar que el navegador se ha bloqueado o se ha colgado el script, si la respuesta del servidor tarda mucho
- Hay que poner el tercer parámetro del método open a **true** (la petición sí es asíncrona)
- El servidor nos avisará "llamando a la función que le designemos" (función "callback")
  - En la propiedad onreadystatechange del XMLHttpRequest "apuntaremos" a la función
  - El callback no se llamará una sola vez, sino varias. El servidor nos informa en la propiedad readyState de XMLHttpRequest de si la respuesta ha empezado a llegar (solo cabeceras HTTP: readyState==2), está cargándose (readyState==3) o completa (readyState==4).

Ejemplo de código asíncrono

- Supongamos que en el servidor esperan un parámetro "cod" con un valor numérico que tenemos en la variable Javascript "codigo".

```
//crear el objeto "xmlhttprequest", versión simplificada que no iría en IE6
var req = new XMLHttpRequest();
//preparar la petición. El tercer parámetro indica que sí es asíncrona
req.open('GET', 'http://www.miservidor.com/miprograma.php?cod=' +codigo, true);
//decir qué función hace de "callback". Esto no se debe hacer antes del open
req.onreadystatechange = mi_callback;
req.send(null);
//ya podemos seguir con otras cosas, este send no se bloquea
```

```
//en algún sitio del javascript debe estar definida esta función
function mi_callback() {
      if (req.readyState == 4) {   //también valdría this.readyState, aunque esto no parece
            if(req.status == 200)  //estar documentado (idem this.status y this.responseText)
                  alert(req.responseText);
}…
```

responseText

- **responseText** es la información que nos envía el servidor, en un String.
- El servidor nos tendrá que enviar la información en un formato adecuado que podamos parsear desde Javascript
  - Por ejemplo, en un chat nos podría mandar los nuevos mensajes recibidos simplemente poniendo un mensaje en cada línea y separando sus datos (hora, usuario, texto) con algún carácter especial
  - Ya veremos cómo se hace desde el servidor para generar esta información, por el momento supongamos que lo hace

```
10:00:05#pepito#"hola a todos…"
10:00:15#jorgito#"hola pepito, cuánto tiempo sin saber de ti! :)"
```

¿Pero dónde está XML?

- En la idea de AJAX original, la información se enviaría en XML, que permite estructurar la información de manera más "elegante" que un formato "casero" ad-hoc

```
<mensajes>
  <mensaje>
    <hora>10:00:05</hora>
    <login>Pepito</login>
    <texto>Hola a todos…</texto>
  </mensaje>
  …
</mensajes>
```

- Problema: aunque el API DOM de Javascript permite parsear XML, resulta tedioso de usar, como veremos en semanas posteriores.

JSON y AJAX

- Si el servidor nos envía JSON, mediante **eval** podemos obtener un objeto Javascript en un solo paso, mientras que el XML tendríamos que "parsearlo"
  - Si desde el servidor nos llega

```
[
    {hora:"10:00:05",login:"pepito",texto:"hola a todos…"},
    {hora:"10:00:15",login:"jorgito",texto:"hola pepito, cuánto
            tiempo sin saber de ti! :)"}
]
```

    - Desde el cliente podemos obtener un array de objetos Javascript (cada uno con los campos hora, login y texto) sin más que hacer

```
mensajes = eval( "(" + xhr.responseText + ")")
```

    - (Suponiendo que xhr es el objeto XMLHttpRequest con el que hemos hecho la petición AJAX)
- Problema de **eval**: evalúa cualquier código Javascript, no solo JSON (posible código "malicioso"). En su lugar podemos usar algo que solo interprete JSON, por ejemplo https://github.com/douglascrockford/JSON-js

Restricciones de seguridad

- **Política de seguridad del "mismo origen"**: un XMLHttpRequest solo puede hacer una petición AJAX al mismo host del que vino la página en la que está definido
  - Resumiendo, una página que tenéis en localhost y que tiene Javascript no puede hacer peticiones AJAX a Google, por ejemplo
- El "cross-domain AJAX" permite romper esta política bajo ciertas circunstancias
  - Si el servidor al que le haces la petición la permite (enviando la cabecera Access-Control-Allow-Origin), el navegador también dejará que se haga

```
HTTP/1.1 200 OK
Server: Apache/2.0.61
Access-Control-Allow-Origin: *
```

AJAX “nivel 2”

- “Versión 2.0” de XMLHttpRequest que permite
  - Intercambiar datos binarios con el servidor. Por ejemplo, esto puede servir para subir archivos
  - Acceder a ciertos eventos, por ejemplo para ir monitorizando el progreso en el envío/recepción de datos
- Solo funciona en navegadores modernos. No en IE9. Consultar caniuse.com u otro recurso equivalente
- API
  - FormData representa los campos de un formulario, incluyendo type=”file”. Enviando el FormData enviamos también el archivo
  - Hay varios eventos como “progress”, “load”,”error” o “abort”
  - La gestión de los eventos se hace con el estándar W3C de event listeners, sobre el objeto XMLHttpRequest

Ejemplo de AJAX 2


```
<script>
        function verProgreso(e) {
            var p= document.getElementById("progreso");
             p.innerHTML = Math.round((e.loaded / e.total)*100)+"%";
        }
        function uploadAJAX() {
            var fdata = new FormData(document.getElementById("formu"))
            var xhr = new XMLHttpRequest();
            xhr.upload.addEventListener("progress", verProgreso, false)
            xhr.open("POST", "Upload", true)
            xhr.onreadystatechange = function() {if (this.readyState==4) alert(this.responseText)}
            xhr.send(fdata)
        }
</script>
...
<form enctype="multipart/form-data" id="formu">
    Elegir archivo: <input type="file" name="archivo"/> <br/>
    <input type="button" value="enviar" onclick="uploadAJAX()"/>
</form>
<p id="progreso"></p>
```