

TEMA 7: MECANISMOS DE COMUNICACIÓN ASÍNCRONA (AJAX)

Desarrollo web en entorno cliente

Objetivos

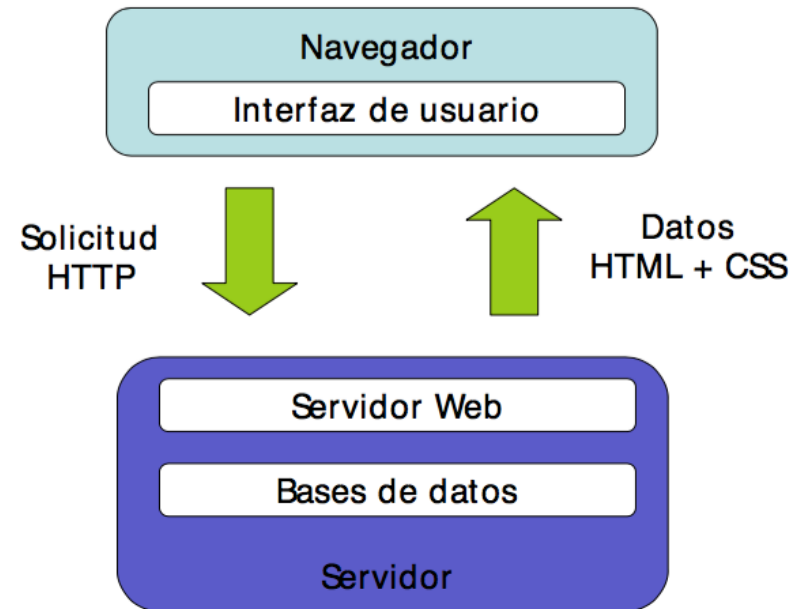
2

- ❑ Mecanismos de comunicación asíncrona en las aplicaciones Web.
- ❑ Tecnologías asociadas con la técnica AJAX.
- ❑ Formatos de envío y recepción de información asíncrona.
- ❑ Llamadas asíncronas.
- ❑ Librerías de actualización dinámicas actuales.

Mecanismos de comunicación síncrona

3

□ En un proceso habitual, el cliente es el que inicia el intercambio de información solicitando datos al servidor que responde enviando un flujo de datos al cliente.



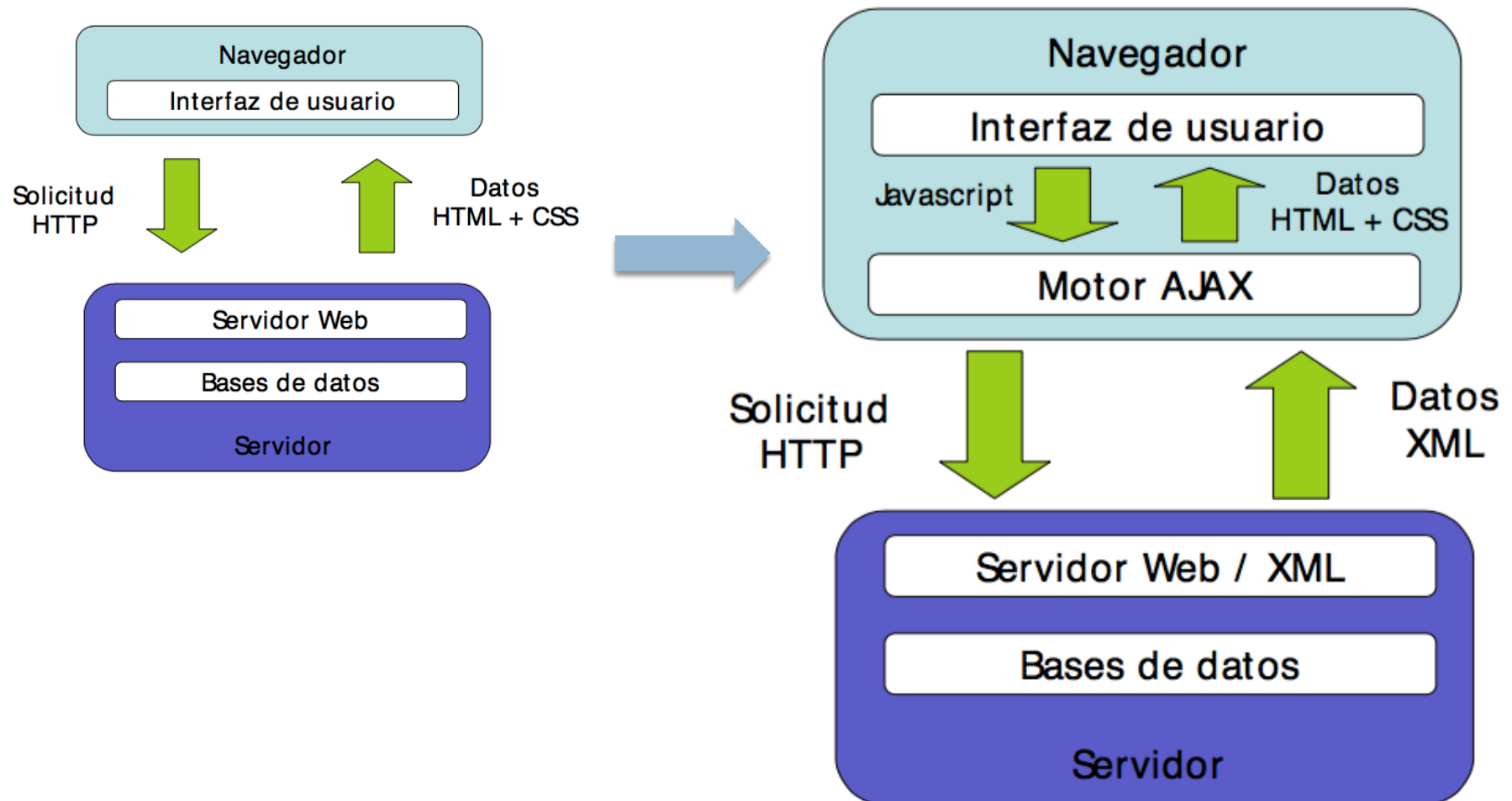
Mecanismos de comunicación asíncrona

4

- El mecanismo de comunicación **asíncrona** recarga en **segundo plano** una **parte** de la página Web, dejando **desbloqueado** el resto.
- El cliente que envía una petición no permanece bloqueado esperando la respuesta del servidor.
- Esto ayuda a que las aplicaciones Web tengan una interactividad similar a las aplicaciones de escritorio.

Mecanismos de comunicación asíncrona (II)

5



AJAX

6

- ¿Por qué? = aplicaciones Web interactivas.
- Solución = nuevo uso de tecnologías como XML, CSS o DOM.
- En 2005 J.J. Garrett habla por primera vez sobre AJAX (*Asynchronous JavaScript And XML*) = **JavaScript asíncrono y XML**.
- Se suprimen los efectos secundarios de las recargas, como la pérdida del contexto, la ubicación del *scroll* o las respuestas más lentas.



Consideraciones al usar AJAX

7

- Uso de nuevas tecnologías y mayor complejidad.
- Las aplicaciones Web o sitios Web con AJAX utilizan más recursos del servidor.
- El uso de las tecnologías asociadas con AJAX no están presentes por defecto en cualquier tipo de dispositivos.
- Aunque cada vez menos, todavía existen incompatibilidades entre navegadores.

Tecnologías utilizadas en AJAX

8

- XHTML y CSS para una presentación basada en estándares.
- DOM para la interacción y la visualización dinámica de datos.
- XML y XSLT* para el intercambio y transformación de datos.
- XMLHttpRequest para la recuperación asíncrona de los datos.
- y JavaScript como elemento de unión.

(*) XSLT (transformaciones XSL) es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML.

Tecnologías involucradas: XHTML y CSS

- XHTML es un HTML estándar especificado mediante un documento XML. XHTML es más riguroso con su estructura que HTML:
 - ▣ Los valores de los atributos, siempre entre comillas:
 - Incorrecto: `<td colspan=2>`.
 - Correcto: `<td colspan="2">`.
 - ▣ Los nombres de elementos y atributos deben ir en minúsculas:
 - Incorrecto: ``.
 - Correcto: ``.
 - ▣ No está permitida la minimización de atributos (se usa el nombre del atributo como valor):
 - Incorrecto: `<textarea readonly>`.
 - Correcto: `<textarea readonly="readonly">`.

XML

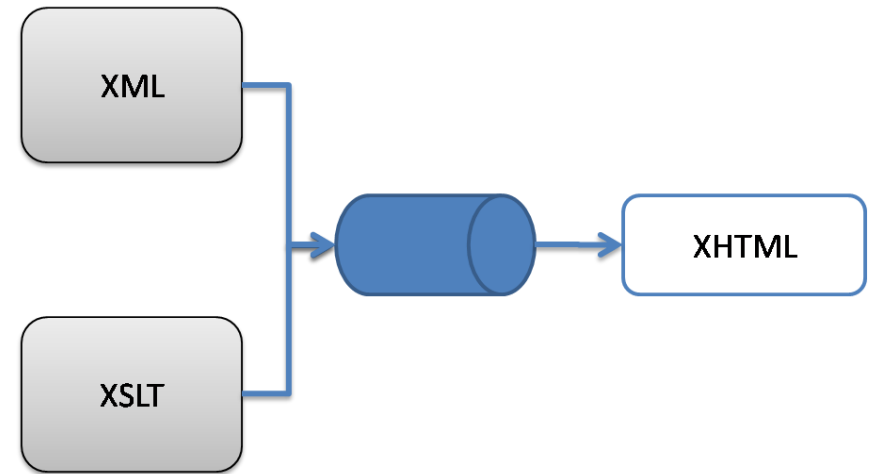
10

- El lenguaje XML es utilizado para describir y estructurar datos.
- Alrededor de XML encontramos otras tecnologías: XQuery, XSLT.
- Los navegadores contienen funcionalidades internas para trabajar con documentos XML.

XSLT

11

- Lenguaje para transformar un documento XML en otro documento XML, texto plano, etc.
- Con XSLT se podría recibir un documento XML representando una tabla, y transformarlo en una tabla XHTML.
- XSLT utiliza otro lenguaje, el XPath, para realizar consultas en el documento XML cuando se aplican las transformaciones.
- En este caso, XPath busca elementos dentro del XML de entrada y XSLT los procesa.



Objeto XmlHttpRequest

12

- Aparece a partir de Internet Explorer 5 en la forma de un control ActiveX llamado XMLHttp.
- Se fueron transformando en un estándar de facto en navegadores como Firefox, Safari y Opera (pero todavía existen diferencias).
- Actualmente el objeto `XMLHttpRequest` se encuentra descrito por el World Wide Web Consortium y sirve como una interfaz con la que se realizan peticiones a servidores Web.

Atributos del objeto XMLHttpRequest

13

Atributo	Descripción
readyState	Devuelve el estado del objeto como sigue: 0 = sin inicializar, 1 = abierto, 2 = cabeceras recibidas, 3 = cargando y 4 = completado.
responseBody	Devuelve la respuesta como un array de bytes.
responseText	Devuelve la respuesta como una cadena.
responseXML	Devuelve la respuesta como XML. Esta propiedad devuelve un objeto documento XML, que puede ser examinado usando las propiedades y métodos del árbol DOM.
status	Devuelve el estado como un número (p. ej. 404 para "Not Found").
statusText	Devuelve el estado como una cadena (p. ej. "Not Found").

Atributos del objeto XMLHttpRequest

14

Atributo	Descripción
status	El atributo statusText devuelve el código del estado HTTP de la transmisión devuelto por el servidor web.
100 Continua	405 Método no permitido
101 Cambio de protocolo	406 No aceptable
200 OK	407 Proxy requerido
201 Creado	408 Tiempo de espera agotado
202 Aceptado	409 Conflicto
203 Información no oficial	410 No mapas disponible
204 Sin Contenido	411 Requiere longitud
205 Contenido para reset	412 Falló precondition
206 Contenido parcial	413 Entidad de solicitud demasiado larga
300 Múltiples posibilidades	414 URI de solicitud demasiado largo
301 Mudado permanentemente	415 Tipo de medio no soportado
302 Encontrado	416 Rango solicitado no disponible
303 Vea otros	417 Falló expectativa
304 No modificado	500 Error interno
305 Utilice un proxy	501 No implementado
307 Redirección temporal	502 Pasarela incorrecta
400 Solicitud incorrecta	503 Servicio no disponible
401 No autorizado	504 Tiempo de espera de la pasarela agotado
402 Pago requerido	505 Versión de HTTP no soportada
403 Prohibido	
404 No encontrado	

Métodos del objeto XMLHttpRequest

15

Métodos	Descripción
<code>abort()</code>	Cancela la petición en curso
<code>getAllResponseHeaders()</code>	Devuelve el conjunto de cabeceras HTTP como una cadena.
<code>getResponseHeader(cabecera)</code>	Devuelve el valor de la cabecera HTTP especificada.
<code>open(método, URL [, asíncrono [, nombreUsuario [, clave]]])</code>	<p>Especifica el método, URL y otros atributos opcionales de una petición.</p> <p>El parámetro de método puede tomar los valores "GET", "POST", o "PUT".</p> <p>El parámetro URL puede ser una URL relativa o completa.</p> <p>El parámetro asíncrono especifica si la petición será gestionada asincrónicamente o no.</p>
<code>send([datos])</code>	Envía la petición con los datos pasados por parámetro como cuerpo de la petición.

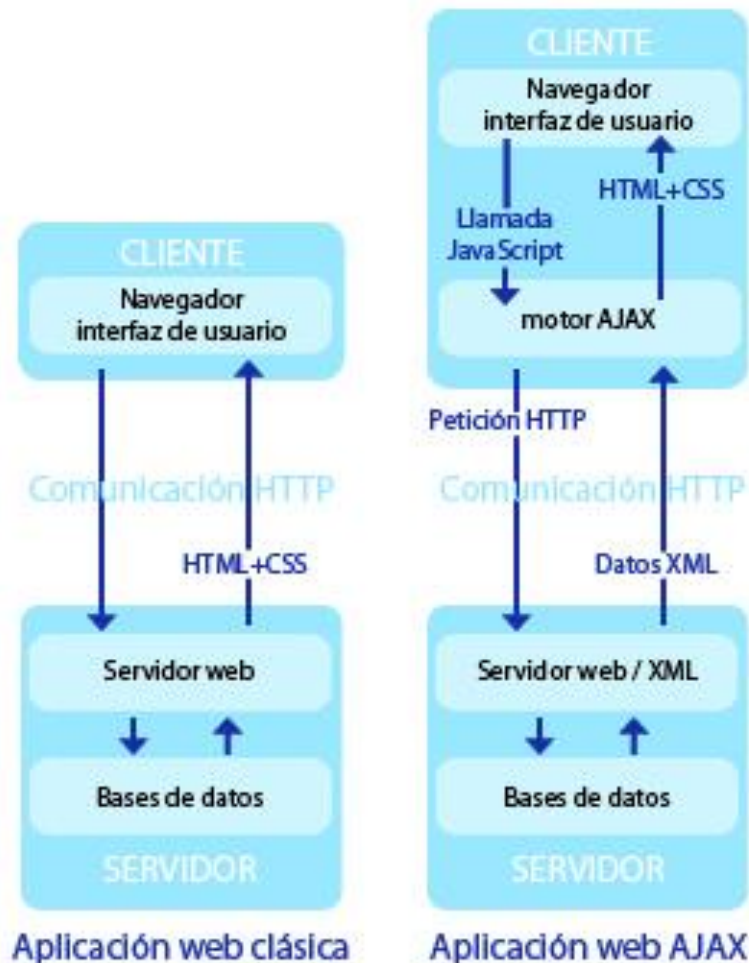
Eventos del objeto XMLHttpRequest

16

Evento	Descripción
<code>onreadystatechange</code>	Evento que se dispara con cada cambio de estado.
<code>onabort</code>	Evento que se dispara al abortar la operación.
<code>onload</code>	Evento que se dispara al completar la carga.
<code>onloadstart</code>	Evento que se dispara al iniciar la carga.
<code>onprogress</code>	Evento que se dispara periódicamente con información de estado.

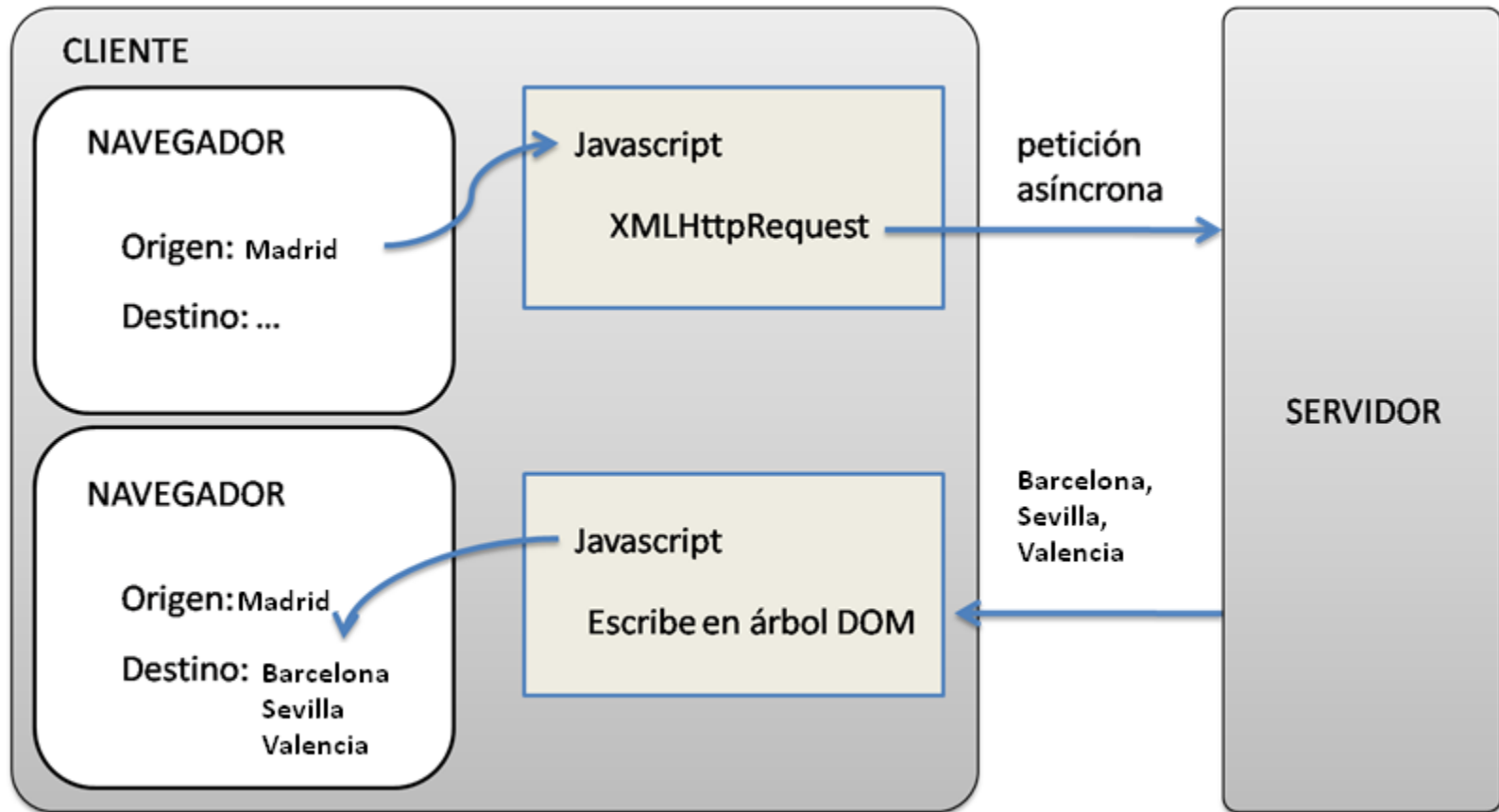
Modelo clásico de aplicaciones Web vs Modelo Ajax de aplicaciones Web

17



Perspectiva Global con AJAX: Ejemplo

18



Ejemplo AJAX

19

- La página Web muestra un botón el cual, cuando hacemos click sobre él, muestra un mensaje en un elemento div cambiando el texto que se encontraba anteriormente.

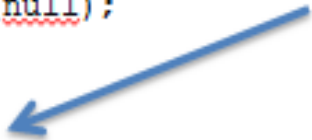
```
<form>
  <input type = "button" value = "Buscar información"
  onclick = "obtenerDatosServidor('http://web/datos.txt',
  'elemento_destino')">
</form>
<div id="elemento_destino">
  <p>La información aparecerá aquí</p>
</div>
```

Ejemplo AJAX

20

1. Función
“obtenerDatosServidor”
contiene dos parámetros.
2. Se elige el elemento HTML
a ser modificado.
3. Se configura una conexión
asíncrona con una URL.
4. Se indica la función a ser
llamada una vez cambie el
estado del objeto XHR.
5. Se abre la conexión.

```
<script language = "javascript">  
    var objetoXHR = new XMLHttpRequest();  
  
    1 function obtenerDatosServidor(origen, elemento)  
    {  
    2     var objeto_destino = document.getElementById(elemento);  
    3     objetoXHR.open("GET", origen);  
    4     objetoXHR.onreadystatechange = respuesta();  
    5     objetoXHR.send(null);  
    }  
    }  
  
    function respuesta(){  
        if (objetoXHR.readyState == 4 &&  
            objetoXHR.status == 200) {  
            objeto_destino.innerHTML = objetoXHR.responseText;  
        }  
    }  
    }  
</script>
```



Ejemplo AJAX

21

- Se puede incorporar un código inicial para comprobar el tipo de navegador y, por tanto, instanciar el objeto `XMLHttpRequest` de la manera correcta.
- Para ello tenemos en cuenta que, dependiendo del navegador, el objeto `XMLHttpRequest` pertenecerá a la implementación del mismo navegador y más precisamente formará parte de su elemento `window`.

```
var objetoXHR = false;
if (window.XMLHttpRequest) {
    objetoXHR = new XMLHttpRequest();
} else if (window.ActiveXObject) { // o directamente else
    objetoXHR = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Función callback

22

- Una función de callback es una función pasada como parámetro a otra función.
- Si se tiene más de una tarea AJAX en un sitio web, hay que crear una función para ejecutar el objeto XMLHttpRequest y una función de devolución de llamada para cada tarea AJAX.
- La llamada de función debe contener la URL y la función a llamar cuando la respuesta está lista.

Función callback

23

```
<script type="text/javascript">
  loadDoc("url-1", myFunction1);
  loadDoc("url-2", myFunction2);
  function loadDoc(url, cFunction) {
    var xhttp;
    xhttp=new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        cFunction(this);
      }
    };
    xhttp.open("GET", url, true);
    xhttp.send();
  }

  function myFunction1(xhttp) {
    // action goes here
  }
  function myFunction2(xhttp) {
    // action goes here
  }
</script>
```

Procesado de ficheros XML

24

```
<script>
function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            myFunction(this);
        }
    };
    xhttp.open("GET", "cd_catalog.xml", true);
    xhttp.send();
}
function myFunction(xml) {
    var i;
    var xmlDoc = xml.responseXML;
    var table="<tr><th>Artist</th><th>Title</th></tr>";
    var x = xmlDoc.getElementsByTagName("CD");
    for (i = 0; i <x.length; i++) {
        table += "<tr><td>" +
            x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
            "</td><td>" +
            x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
            "</td></tr>";
    }
    document.getElementById("demo").innerHTML = table;
}
</script>
```


Llamada AJAX con POST

25

```
function buscar() {  
    var dni = document.getElementById("id_dni").value;  
    var pag = "ajax7.php";  
    datos = "dni=" + encodeURIComponent(dni) + "&nocache=" + Math.random();  
    loadDoc(pag, datos, mostrar);  
}  
  
function loadDoc(url, data, cFunction) {  
    var xhttp;  
    xhttp = nuevoAjax();  
    xhttp.onreadystatechange = function () {  
        if (this.readyState == 4 && this.status == 200) {  
            cFunction(this);  
        }  
    };  
    xhttp.open("POST", url, true);  
    xhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  
    xhttp.send(data);  
}
```

Llamada AJAX con POST

26

- `xhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");`
 - si no se establece la cabecera Content-Type correcta, el servidor descarta todos los datos enviados mediante el método POST.
 - para enviar parámetros mediante el método POST, es obligatorio incluir la cabecera Content-Type
- `xhttp.send(data);`
 - se encarga de enviar los parámetros al servidor (cadena de texto, documento XML,...)
 - Los parámetros se envían en forma de cadena:
 - variables = valores, concatenados mediante `&`
 - Recomendación: añadir como último parámetro "nocache" conteniendo un número aleatorio creado mediante el método `Math.random()`, como estrategia para evitar problemas con la caché de los navegadores. Como en cada petición varía al menos el valor de uno de los parámetros, el navegador está obligado siempre a realizar la petición directamente al servidor y no utilizar su cache. Se puede utilizar en peticiones GET y POST.
 - `encodeURIComponent()`: imprescindible para evitar problemas con caracteres especiales. Se debe utilizar para codificar un URI Component (un string que supuestamente debe ser parte de una URL).

Librerías de Actualización Dinámica

27

- Junto con la tecnología AJAX hay librerías como JQuery o Prototype, que implementan una gran cantidad de funciones y controles a ser utilizados por los desarrolladores, y que permiten hacer peticiones AJAX de forma más amigable:
 - ▣ Independiente de la tecnología del servidor (por ejemplo PHP, JSP, ASP, etc.).
 - ▣ Manejar de manera transparente las incompatibilidades de los diferentes navegadores.
 - ▣ Manejar la comunicación asíncrona, sin necesidad de realizar la gestión de las operaciones de bajo nivel, como por ejemplo el manejo de estados y de tipos de errores.
 - ▣ Acceso sencillo al árbol DOM.
 - ▣ Información de errores para facilitar su utilización al desarrollador.
 - ▣ Proporcionar controles y objetos gráficos configurables, como por ejemplo: botones, calendarios, campos de texto.

Fetch API

28

- Fetch API es una forma alternativa de realizar peticiones AJAX similares al objeto XMLHttpRequest.
- **Fetch** API se basa en el uso de promesas JavaScript, por lo que resulta necesario familiarizarse primero con ellas antes de utilizar este API.
- El uso de **promesas** permite un código más claro, limpio y sencillo, que mediante el uso del objeto XMLHttpRequest.
- Sin embargo, Fetch API tiene ciertas limitaciones que no tiene el objeto XMLHttpRequest, como puede ser la imposibilidad de monitorizar el progreso de la petición AJAX (útil para cuando transferimos un archivo grande al servidor, por ejemplo), o la imposibilidad de abortar/cancelar una petición AJAX.

JSON

29

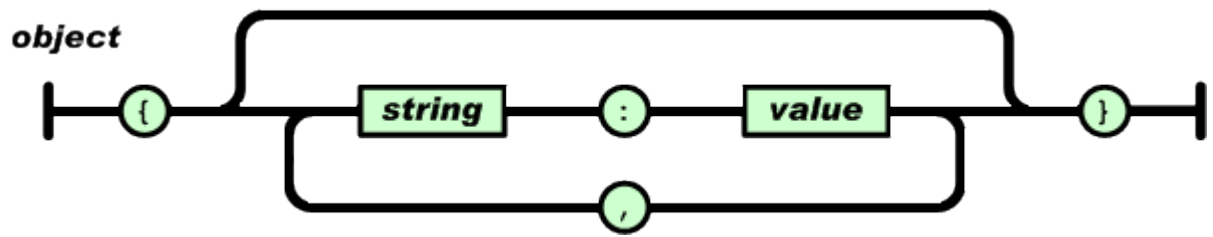
- JSON (JavaScript Object Notation).
- Formato ligero para el intercambio de datos.
- Es una manera de almacenar información.
- Fue pensado en un primer momento como una alternativa a XML (tiene una gran cantidad de información extra asociada a su estructura).

JSON

30

- JSON está constituido por dos estructuras:
 - ▣ Una colección de pares de nombre/valor.
 - En varios lenguajes esto es conocido como un *objeto*, *registro*, ...
 - ▣ Una lista ordenada de valores.
 - En la mayoría de los lenguajes, esto se implementa como arreglos.

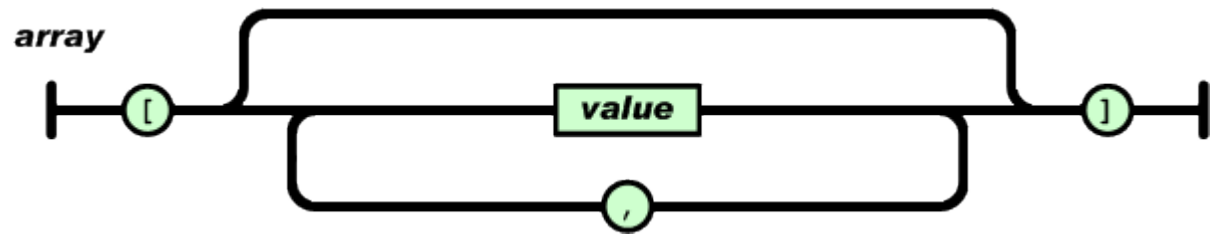
JSON



31

- El elemento base de la sintaxis es el *object*.
- Está conformado por un conjunto desordenado de pares nombre/valor.
- Un objeto comienza con una llave de apertura y finaliza con una llave de cierre.
- Cada nombre es seguido por dos puntos, estando los pares nombre/valor separados por una coma.

JSON



32

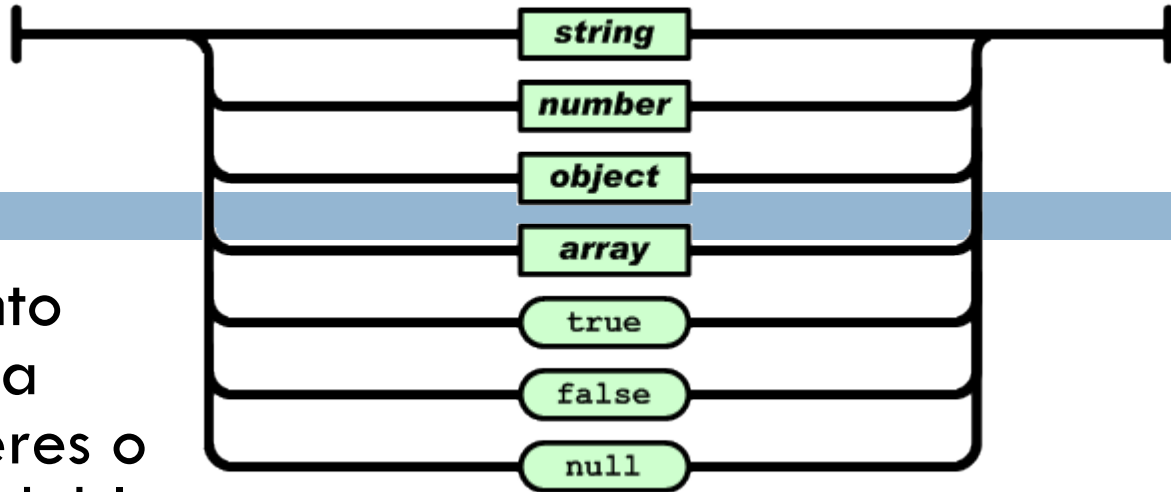
- Un *array* es una colección de elementos *values*.
- Comienza por un corchete izquierdo y termina con un corchete derecho.
- Los elementos *value* se separan por una coma.

JSON

33

- A su vez un elemento *value* puede ser una cadena de caracteres o *string* con comillas dobles, un *number*, los valores booleanos *true* o *false*, *null*, un *object* o un *array*.
- Estas estructuras pueden anidarse.

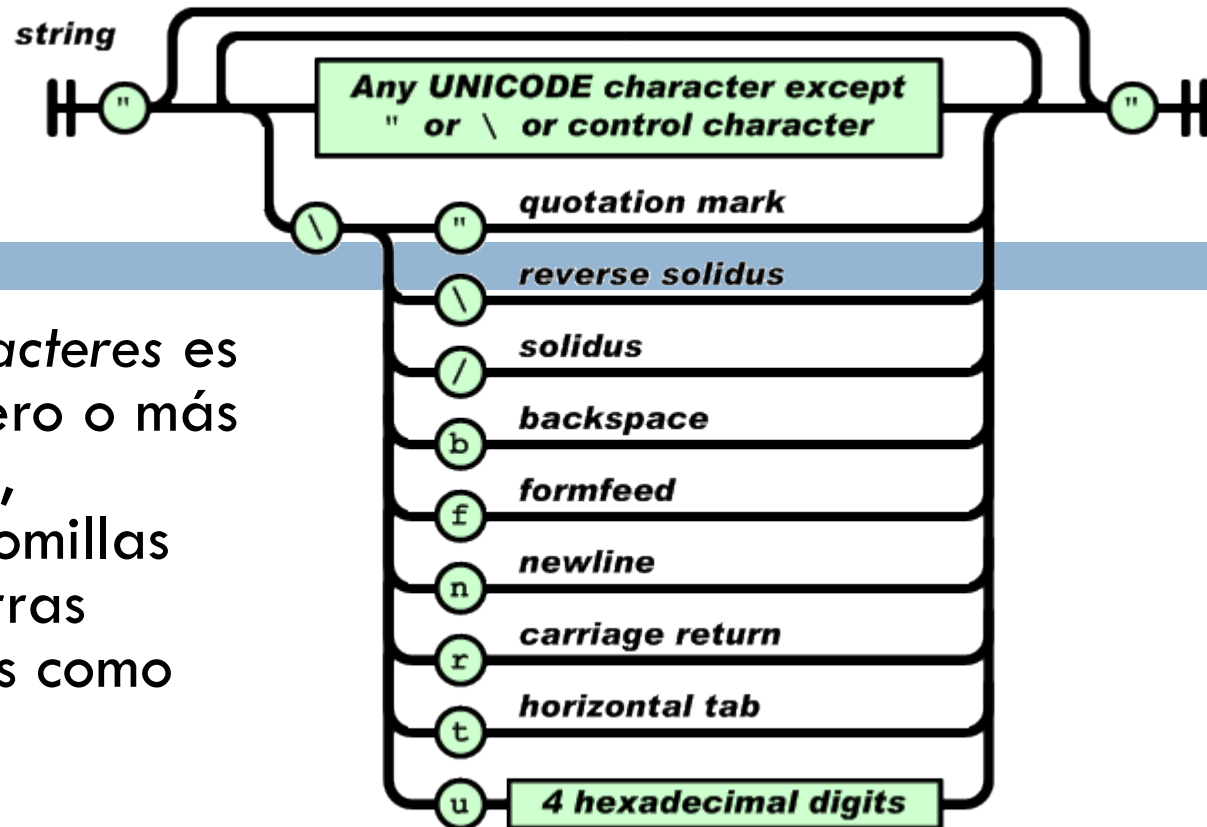
value



JSON

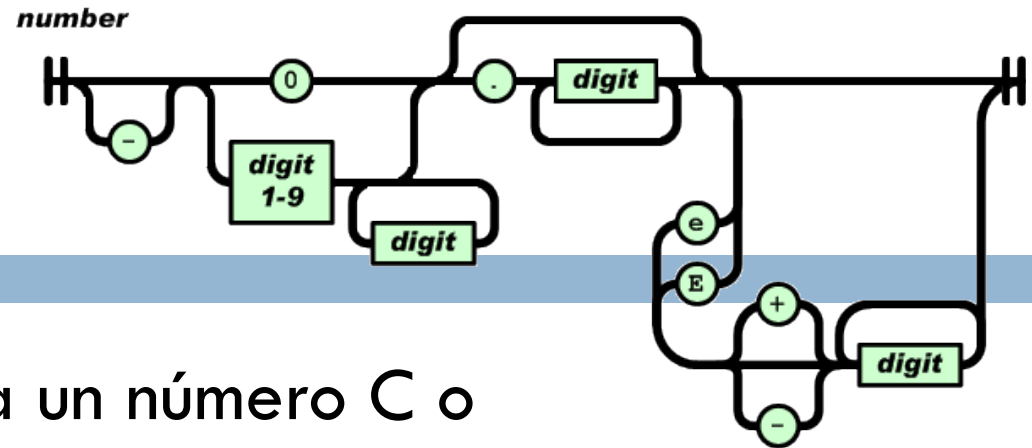
34

- Una *cadena de caracteres* es una colección de cero o más caracteres Unicode, encerrados entre comillas dobles, usando barras divisorias invertidas como escape.
- Un carácter está representado por una cadena de caracteres de un único carácter.
- Una *cadena de caracteres* es parecida a una cadena de caracteres C o Java.



JSON

35



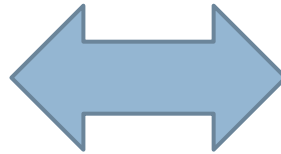
- Un *número* es similar a un número C o Java, excepto que no se usan los formatos octales y hexadecimales.
- Los espacios en blanco pueden insertarse entre cualquier par de símbolos.
- Exceptuando pequeños detalles de *encoding*, esto describe completamente el lenguaje.

JSON

36

```
{  
  'nombre': 'pepe',  
  'edad': 34,  
  'domicilio': 'calle alcalá 1',  
  'estudios': ['primario',  
               'secundario',  
               'universitario']  
}
```

JSON



```
<ciudadano>  
  <nombre>pepe</nombre>  
  <edad>34</edad>  
  <domicilio>  
    calle alcalá 1  
  </domicilio>  
  <estudios>  
    <estudio>primario</estudio>  
    <estudio>secundario</estudio>  
    <estudio>universitario</estudio>  
  </estudios>  
</ciudadano>
```

XML

JSON

37

□ Array literal en JavaScript:

```
var usuario=["juan","26"]; //usuario[0]
```

□ Objeto literal en JavaScript:

```
var persona={    // persona.nombre  persona.clave  persona.edad
  "nombre":"juan",
  "clave":"xyz",
  "edad":"26"
};
```

□ Combinación de ambos:

```
var persona={
  "nombre":"juan",
  "edad":"22",
  "estudios":["primario","secundario"]
};
```

JSON



```
{
  "nombre":"juan",
  "edad":"22",
  "estudios":["primario","secundario"]
}
```

JSON

38

```
function presionBoton(e)
{
    var cadena='{ "microprocesador": "pentium", ' +
                '"memoria": 1024, ' +
                '"discos": [80, 250] ' +
                '}';
    var maquina=JSON.parse(cadena);
    window.alert('microprocesador: '+maquina.microprocesador);
    window.alert('Memoria ram: '+maquina.memoria);
    window.alert('Capacidad disco 1: '+maquina.discos[0]);
    window.alert('Capacidad disco 2: '+maquina.discos[1]);
}
```

- **JSON.parse(cadena)** convierte una cadena en formato JSON a un objeto JavaScript.
- **JSON.stringify(objeto)** convierte un objeto de JavaScript en una cadena de texto JSON