



JavaScript

PROCESOS ASÍNCRONOS DE COMUNICACIÓN CON EL SERVIDOR

Comunicación síncrona



La comunicación síncrona implica un cliente que espera que el servidor responda a un mensaje.



Los mensajes pueden fluir en ambas direcciones.



El remitente envía un mensaje al receptor y el receptor recibe este mensaje y da respuesta al remitente.



El remitente no enviará otro mensaje hasta que reciba una respuesta del receptor.



Por ejemplo:

Cuando se carga una página

Al hacer clic en un enlace

Cuando se cumplimenta un formulario de login, se envían los datos y hay que esperar a que el servidor envíe la respuesta (que será la misma página si hay error u otra si es exitosa)

Comunicación asíncrona

- El cliente puede realizar varias peticiones al servidor sin necesidad de esperar por la respuesta de la primera.
- Permite recargar 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. Ayuda a que las aplicaciones web tengan una interactividad similar a las aplicaciones de escritorio y es en parte lo que hace algunos años se denomina Web 2.0.
- Ejemplos de uso:
 - Valorar una noticia con estrellas o una noticia del muro de Facebook con el “me gusta”
 - Borrar un correo en gmail
 - Comentar una noticia

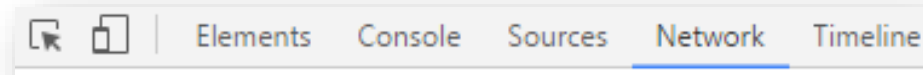
A horizontal yellow bar spanning the width of the slide, with a thin grey line above it.

Examinando procesos síncronos y asíncronos

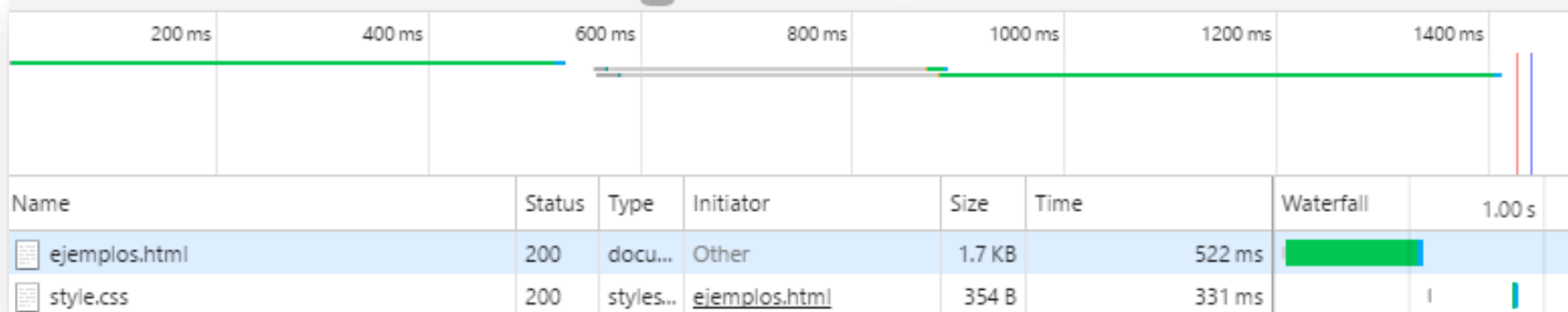
La herramienta de Network del navegador



- Accede a una página cualquiera en tu navegador
- Accede a la consola de desarrollador de tu navegador (F12)
- Ve a la pestaña network




- Recarga la página (F5)
- Observa las peticiones que se han realizado al servidor en la línea de tiempo



Observando procesos síncronos



- Con la herramienta Network activa
- Accede a <https://github.com/login>
- Observa la línea de tiempo
- Limpia el log con:

- Introduce unos datos cualquiera para acceder
- Observa la línea de tiempo y cómo la página se recarga

Observando procesos asíncronos



- Con la herramienta Network activa
- Accede a <https://mail.google.com>
- Observa la línea de tiempo
- Limpia el log con:



- Marca alguno de los mensajes como importante (estrella)
- Observa la línea de tiempo, cómo se envía comunicación al servidor y cómo la página NO se recarga
- Desmarca alguno de los mensajes como importante (quita la estrella) y observa



AJAX

AJAX

- AJAX es el acrónimo de *Asynchronous Javascript and XML*, es decir, Javascript y XML Asíncrono
- Es una tecnología que permite la comunicación asíncrona entre un servidor y un navegador en formato XML mediante programas escritos en Javascript.
- El principal objetivo del AJAX, es intercambiar información entre el servidor y el cliente (navegadores) sin la necesidad de recargar la página. De esta forma, ganamos en usabilidad, experiencia y productividad del usuario final.



Ventajas

- Rapidez en las operaciones.
- Menos carga del servidor (menos transferencia de datos cliente/servidor).
- Menos ancho de banda.
- Soportada por la mayoría de navegadores.
- Interactividad (El usuario no tiene que esperar hasta que lleguen los datos del servidor).
- Portabilidad, Usabilidad
- Velocidad (Debido a que no hay que recargar la página nuevamente)

Limitaciones

- Se pierde el concepto de “volver a la página anterior”.
- Problemas con navegadores antiguos.
- No funciona si el usuario tiene desactivado el Javascript en su navegador.
- Se requieren conocimiento sobre las tecnologías que forman AJAX.
- Problemas SEO, los buscadores no indexan la información recibida vía AJAX.

Proceso de una comunicación Ajax

1. El usuario provoca un evento.
2. Se crea y configura un objeto XMLHttpRequest.
3. El objeto XMLHttpRequest realiza una llamada al servidor.
4. La petición se procesa en el servidor.
5. El servidor retorna un documento XML que contienen el resultado.
6. El objeto XMLHttpRequest llama a la función callback() y procesa el resultado.
7. Se actualiza el DOM de la página asociado con la petición con el resultado devuelto.



El objeto XMLHttpRequest - XHR

- Proporciona los métodos y propiedades necesarias para la comunicación con el servidor mediante el protocolo HTTP.
- La comunicación puede ser realizada con los métodos GET/POST. La respuesta puede ser de tipo plano/xml.
- Trabaja en background y existe un número limitado de peticiones.
- Permite especificar un manejador para el control de cambios de estado.
- El manejador notifica del estado de la petición con los estados:
 - Inicializada: código 1
 - Iniciada: código 2
 - En proceso: código 3
 - Operación completada: código 4

Ejemplo

```
let req = new XMLHttpRequest();
```

Creamos el objeto AJAX

```
req.onreadystatechange =  
metodoCallback;
```

Programamos la función de
callback cuando esté lista la
respuesta

```
req.open("GET", "URL", true);  
req.send();
```

Indicamos el método de
petición y la URL a abrir

Enviamos la petición

...

```
function metodoCallback() {  
  if (req.readyState == 4) {  
    doSomethingWith(req.responseText);  
  } else if (req.readyState == 3) {  
    showProgressIndicator();  
  }  
}
```

Si está listo (código 4),
procesamos la respuesta

Si NO está listo (en progreso),
mostramos un indicador de
carga

Funciones del objeto XMLHttpRequest

- **open**("method", "URL", syn/asyn): Asigna la URL de destino, el método y otros parámetros opcionales de una petición pendiente
- **send**(content): Envía la petición, opcionalmente se puede enviar una cadena de texto o un objeto DOM
- **abort**(): Detiene la petición actual
- **getAllResponseHeaders**(): Devuelve todas las cabeceras de la respuesta como pares de etiqueta y valores en una cadena
- **getResponseHeader**("headerLabel"): Devuelve el valor de una cabecera determinada
- **setRequestHeader**("label", "value"): Asigna un valor al par label/value para la cabecera enviada.

Propiedades del objeto XMLHttpRequest

- **onreadystatechange**: El manejador del evento llamado en cada cambio de estado del objeto
- **readyState**: Indica el estado del objeto o la petición
 - 0 = sin inicializar
 - 1 =cargando
 - 2 = fin de la carga
 - 3 = actualizando la información recibida
 - 4 = Operación completada
- **status**: Estado HTTP devuelto por el servidor
 - 404 si la página no se encuentra
 - 200 si todo ha ido bien
- **responseText**: Cadena de texto con los datos devueltos por el servidor
- **responseXML**: Objeto DOM devuelto por el servidor
- **statusText**: Respuesta del servidor asociada al status (mensaje de texto)

Procesado de la respuesta en el cliente

- Una vez recibida la respuesta del servidor en **responseText** podremos manipularla mediante Javascript, por ejemplo para inyectarla en el HTML

```
document.getElementById('resultado_ajax').innerHTML = req.responseText;
```

- Normalmente usaremos una función de auxiliar para hacer esta manipulación

```
procesaRespuesta(req.responseText);
```

```
function procesaRespuesta(html){  
    document.getElementById("demo").innerHTML =  
    html;  
}
```


Pongámoslo en práctica – carga de contenido AJAX



Página principal



- Crea una página con un botón y bloque div

```
<p><button  
id="btn_ajax">Pedir</button></p>  
<h2>Resultado ajax</h2>  
<div id="resultado_ajax"></div>
```

- El botón traerá un trozo de código html y lo inyectaremos en el div con id “resultado_ajax”

Trozo html a cargar



- Crea una página html (resultado.html) con un texto

```
<li>Hola</li>  
<li>Cómo</li>  
<li>Estás</li>
```

El JS con AJAX



- Añade un archivo javascript con el código para solicitar el contenido de “resultados.html”

```
let req = new XMLHttpRequest();//objeto AJAX

document.getElementById('btn_ajax').onclick = function (evnt) {

    req.open("GET", 'ajax/resultado.html', true); //la dirección que pedimos
    req.onreadystatechange = funcionDeCallback; // la función de callback
    req.send(); //envío de petición

}

function funcionDeCallback(evnt){
    if (req.readyState == 4) { //4:la respuesta ha terminado de llegar
        document.getElementById('resultado_ajax').innerHTML = req.responseText;
    }
}
```

El JS con AJAX



- En tu navegador abre la herramienta de desarrollador (F12) y accede a la pestaña Network
- Carga la página principal
- Haz click en el botón pedir
- Observa la línea de tiempo en Network y el resultado

Pongámoslo en práctica



- Siguiendo el ejemplo anterior, modifica resultado.html para que devuelva el siguiente html

```
<li><span>Este es el botón hola:</span><button id="a1" attr-  
val="Hola">Hola</button></li>  
<li><span>Este es el botón adiós:</span><button id="a2" attr-  
val="Adiós">Adiós</button></li>
```

- En el JS crea una función para que una vez recibida la respuesta asocie a los botones embebidos en el html eventos, tal que al hacer click muestre el valor de “attr-val” en un alert.

API XHR2 (AJAX de segundo nivel)

- XMLHttpRequest de nivel 2 introduce una gran cantidad de nuevas funciones que ponen fin a problemas en aplicaciones web, como solicitudes de origen cruzado, eventos de progreso de subidas y compatibilidad con subida/bajada de datos binarios.
 - <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>
- Esto permite a AJAX trabajar en coordinación con muchas de las API HTML5 más punteras, como API de FileSystem, el API de Web Audio y WebGL.
- En la web de html5rocks, se puede entender un resumen de estas nuevas funcionalidades
 - <https://www.html5rocks.com/es/tutorials/file/xhr2/>

Ejemplo

```
function leerVentas_Nivel_2() {  
    var url = "ventas.json";  
    var request = new XMLHttpRequest();  
  
    // Opcionalmente, indicar el formato de la  
    respuesta.  
    // request.responseType = "application/json";  
    request.open("GET", url);  
    request.onload = function () {  
        if (request.status == 200) {  
            actualizarIU(request.responseText);  
        }  
    };  
  
    request.send(null);  
}
```


API XHR2 - atributos

Attribute	type	Explanation
onloadstart	loadstart	When the request starts.
onprogress	progress	While loading and sending data.
onabort	abort	When the request has been aborted, either by invoking the abort() method or navigating away from the page.
onerror	error	When the request has failed.
onload	load	When the request has successfully completed.
ontimeout	timeout	When the author specified timeout has passed before the request could complete.
onloadend	loadend	When the request has completed, regard

Origen cruzado

- Una limitación de XHR1 fue la política de mismo de origen (mismo dominio)
- Ahora, XMLHttpRequest admite peticiones de origen cruzado, siempre que esté habilitado el intercambio de recursos de origen cruzado (CORS).
 - <http://dvcs.w3.org/hg/cors/raw-file/tip/Overview.html>
- La diferencia crítica es que la URL de destino debe permitir el acceso desde el origen solicitante enviando una cabecera de respuesta **Access-Control-Allow-Origin**.

Pongámoslo en práctica



Modifica los ejercicios anteriores para usar XHR2

API de AJAX Nivel 2 con soporte de datos JSON

- Los datos pueden ser recibidos en varios formatos: JSON, XML, Otros...
- La entidad IANA establece en su página los formatos MIME posibles y los valores reconocidos:
 - <http://www.iana.org/assignments/media-types/application/index.html>
- JSON es un formato de datos estándar, reconocido por la W3C, ideado por Douglas Crockford, para facilitar los intercambios de información.
- Existen librerías para interpretar JSON en multitud de lenguajes, tal y como podemos ver en la página oficial:
 - <http://www.json.org/>

AJAX + JSON

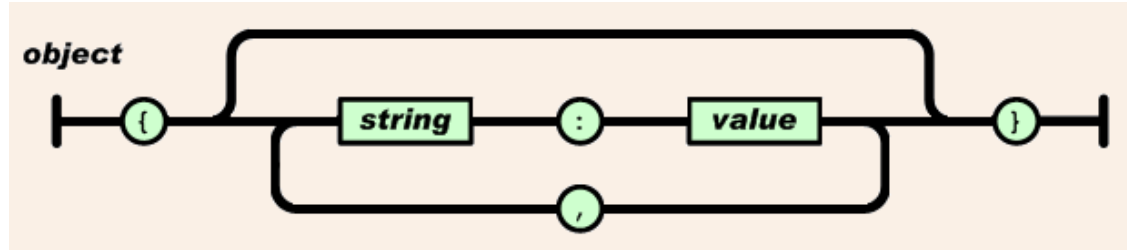
- Los datos JSON (JavaScript Object Notation) tienen un aspecto como este:

```
[{"name":"Valladolid", "TVentas":65292},  
{"name":"Madrid", "TVentas":83444}, ... ]
```

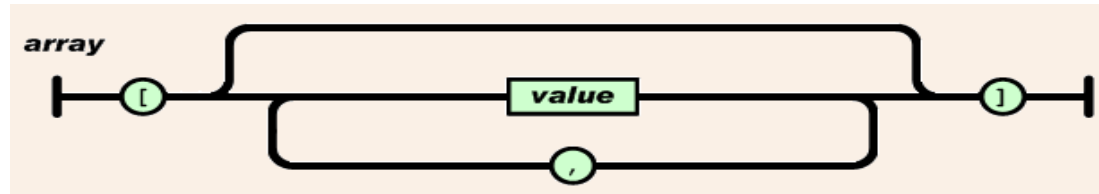
- Donde los nombres de los campos se indican entre comillas dobles, y los valores asociados dependen del tipo de dato.
- <http://json.org/example.html>

Esquemas JSON

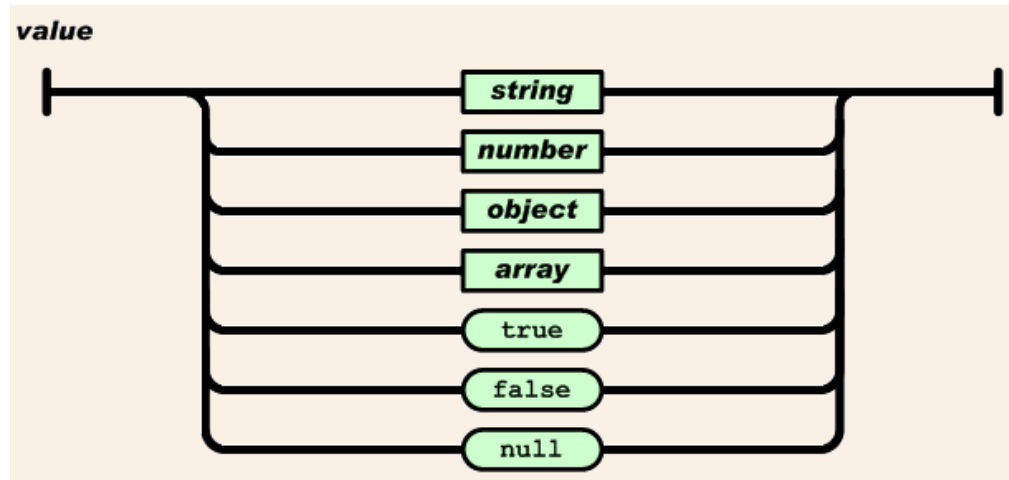
➤ Objetos



➤ Para arrays:



➤ Y para valores:



La respuesta con JSON

- El valor de respuesta incluirá una cabecera como la siguiente, donde se indica el formato y su longitud total, seguido de un bloque de datos formateado:

```
HTTP/1.1 200 OK
Content-length 756
Content-type: application/json
[{"name":"Valladolid","TVentas":65292},
{"name":"Madrid", "TVentas":83444},
ETC...
```

Tipos MIME: <http://www.sitepoint.com/web-foundations/mime-types-complete-list/>

Petición

- Para la petición, utilizaremos típicamente una rutina JavaScript
- Estableceremos cuál es el formato que esperamos en la respuesta y quién va a ser el receptor de los datos

```
request.responseType = "application/json";
```

- Opcionalmente, la propiedad `setRequestHeader()`, permite configurar la petición en la cabecera, usando un formato como este:

```
request.setRequestHeader("Cabecera", "Valor  
asignado");
```


Ejemplo

```
function leerVentas() {  
    var url = "http://localhost:1565/DemosJS5/ventas.json";  
    var request = new XMLHttpRequest();  
    request.responseType = "application/json"; //Predeterminado  
    request.open("GET", url); //Solo configura la llamada  
    request.onload = function() {  
        if (request.status == 200) { //200 significa correcto.  
            actualizarIU(request.responseText);  
        }  
    };  
    request.send(null);  
}
```

Manipulación JSON

- Además, para la manipulación de datos JSON, las implementaciones de JavaScript de los navegadores actuales, incluyen un objeto del mismo nombre, que incorpora métodos útiles para facilitar su manejo:
 - **stringify()** convierte un objeto JavaScript a cadena JSON
 - **parse()** realiza la operación contraria

```
var data =  
JSON.parse(request.responseText);
```

```
var newJSON = JSON.  
stringify(data);
```

Pongámoslo en práctica



- Usa Ajax para cargar JSON en tu página. El JSON devolverá varios parámetros.
- Averigua sus valores
- Leer la función callback y llamarla pasándole los datos json
- En la función de callback
 - Carga el HTML
 - Asociar los botones a eventos onclick que cambien el tamaño del texto en función de las clases predefinidas