JavaScript - Introducción a AJAX

El término AJAX se presentó por primera vez en el artículo "Ajax: A New Approach to Web Applications" publicado por Jesse James Garrett el 18 de Febrero de 2005. Hasta ese momento, no existía un término normalizado que hiciera referencia a un nuevo tipo de aplicación web que estaba apareciendo.

En realidad, el término AJAX es un acrónimo de *Asynchronous JavaScript* + *XML*, que se puede traducir como "JavaScript asíncrono + XML".

El artículo define AJAX de la siguiente forma:

Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes.

Las tecnologías que forman AJAX son:

- XHTML y CSS, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de información.
- XMLHttpRequest, para el intercambio asíncrono de información.
- JavaScript, para unir todas las demás tecnologías.

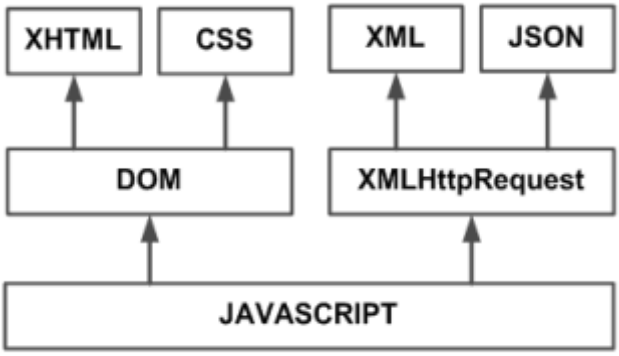

Figura 1.1 Tecnologías agrupadas bajo el concepto de AJAX

Desarrollar aplicaciones AJAX requiere un conocimiento avanzado de todas y cada una de las tecnologías anteriores.

En las aplicaciones web tradicionales, las acciones del usuario en la página (pinchar en un botón, seleccionar un valor de una lista, etc.) desencadenan llamadas al servidor. Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

En el siguiente esquema, la imagen de la izquierda muestra el modelo tradicional de las aplicaciones web. La imagen de la derecha muestra el nuevo modelo propuesto por AJAX:

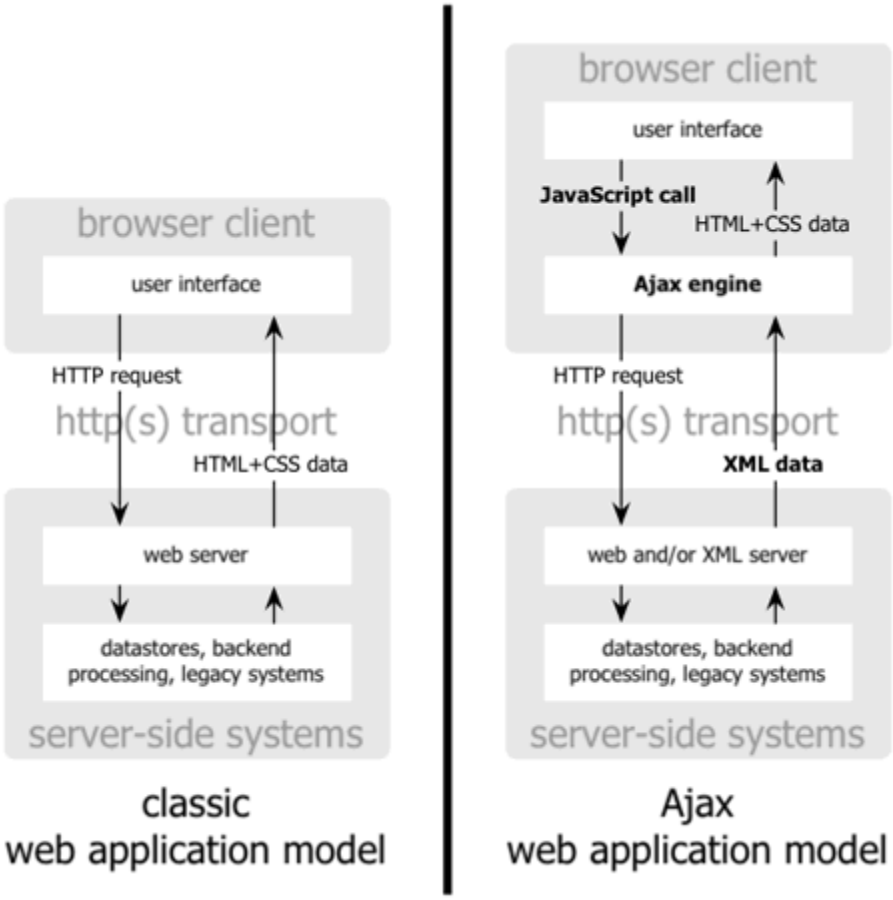

Figura 1.2 Comparación modelo tradicional y nuevo modelo propuesto por AJAX.

Esta técnica tradicional para crear aplicaciones web funciona correctamente, pero no crea una buena sensación al usuario. Al realizar peticiones continuas al servidor, el usuario debe esperar a que se recargue la página con los cambios solicitados. Si la aplicación debe realizar peticiones continuas, su uso se convierte en algo molesto

AJAX permite mejorar completamente la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.

Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor.

El siguiente esquema muestra la diferencia más importante entre una aplicación web tradicional y una aplicación web creada con AJAX. La imagen superior muestra la interación síncrona propia de las aplicaciones web tradicionales. La imagen inferior muestra la comunicación asíncrona de las aplicaciones creadas con AJAX.

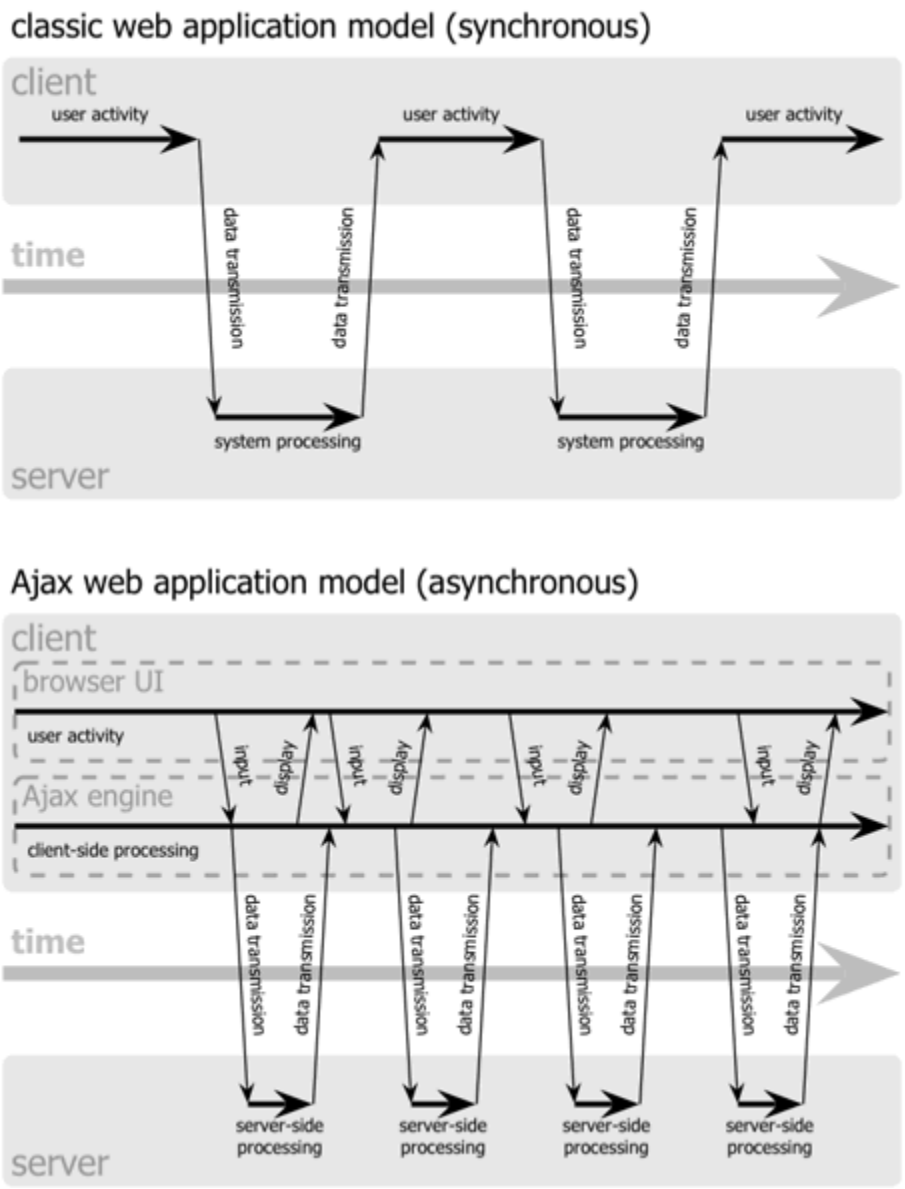

Figura 1.3 Comparación entre comunicaciones síncronas y asíncronas de AJAX

Las peticiones HTTP al servidor se sustituyen por peticiones JavaScript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata. Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de página o largas esperas por la respuesta del servidor.

Primeros pasos con AJAX

Código fuente

La aplicación AJAX completa más sencilla consiste en una adaptación del clásico *"Hola Mundo"*. En este caso, una aplicación JavaScript descarga un archivo del servidor y muestra su contenido sin necesidad de recargar la página.

Código fuente completo:

```
<!DOCTYPE html >
<html>
   <head>
      <title>Hola Mundo con AJAX</title>
      <script type="text/javascript">
         function descargaArchivo() {
            // Obtener la instancia del objeto XMLHttpRequest
   if(window.XMLHttpRequest) {
     peticion_http = new XMLHttpRequest();
   }
   else if(window.ActiveXObject) {
     peticion_http = new ActiveXObject("Microsoft.XMLHTTP");
   }

   // Preparar la funcion de respuesta
   peticion_http.onreadystatechange = muestraContenido;

   // Realizar peticion HTTP
   peticion_http.open('GET', 'http://localhost/holamundo.txt', true);
   peticion_http.send(null);

   function muestraContenido() {
     if(peticion_http.readyState == 4) {
       if(peticion_http.status == 200) {
         alert(peticion_http.responseText);
       }
     }
   }
}

window.onload = descargaArchivo;
      </script>
   </head>
   <body></body>
</html>
```

En el ejemplo anterior, cuando se carga la página se ejecuta el método JavaScript que muestra el contenido de un archivo llamado holamundo.txt que se encuentra en el servidor. La clave del código anterior es que la petición HTTP y la descarga de los contenidos del archivo se realizan sin necesidad de recargar la página.

Análisis detallado

La aplicación AJAX del ejemplo anterior se compone de cuatro grandes bloques: instanciar el objeto XMLHttpRequest, preparar la función de respuesta, realizar la petición al servidor y ejecutar la función de respuesta.

Todas las aplicaciones realizadas con técnicas de AJAX deben instanciar en primer lugar el objeto XMLHttpRequest, que es el objeto clave que permite realizar comunicaciones con el servidor en segundo plano, sin necesidad de recargar las páginas.

La implementación del objeto XMLHttpRequest depende de cada navegador, por lo que es necesario emplear una discriminación sencilla en función del navegador en el que se está ejecutando el código:

```
if(window.XMLHttpRequest) {  // Navegadores que siguen los estándares
  peticion_http = new XMLHttpRequest();
}
else if(window.ActiveXObject) {  // Navegadores obsoletos
  peticion_http = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Los navegadores que siguen los estándares (Firefox, Safari, Opera, Internet Explorer 7 y 8) implementan el objeto XMLHttpRequest de forma nativa, por lo que se puede obtener a través del objeto window. Los navegadores obsoletos (Internet Explorer 6 y anteriores) implementan el objeto XMLHttpRequest como un objeto de tipo ActiveX.

Una vez obtenida la instancia del objeto XMLHttpRequest, se prepara la función que se encarga de procesar la respuesta del servidor. La propiedad onreadystatechange del objeto XMLHttpRequest permite indicar esta función directamente incluyendo su código mediante una función anónima o indicando una referencia a una función independiente. En el ejemplo anterior se indica directamente el nombre de la función:

```
peticion_http.onreadystatechange = muestraContenido;
```

El código anterior indica que cuando la aplicación reciba la respuesta del servidor, se debe ejecutar la función muestraContenido(). Como es habitual, la referencia a la función se indica mediante su nombre sin paréntesis, ya que de otro modo se estaría ejecutando la función y almacenando el valor devuelto en la propiedad onreadystatechange.

Después de preparar la aplicación para la respuesta del servidor, se realiza la petición HTTP al servidor:

```
peticion_http.open('GET', 'http://localhost/prueba.txt', true);
peticion_http.send(null);
```

Las instrucciones anteriores realizan el tipo de petición más sencillo que se puede enviar al servidor. En concreto, se trata de una petición de tipo GET simple que no envía ningún parámetro al servidor. La petición HTTP se crea mediante el método open(), en el que se incluye el tipo de petición (GET), la URL solicitada (http://localhost/prueba.txt) y un tercer parámetro que vale true.

Una vez creada la petición HTTP, se envía al servidor mediante el método send(). Este método incluye un parámetro que en el ejemplo anterior vale null. Más adelante se ven en detalle todos los métodos y propiedades que permiten hacer las peticiones al servidor.

Por último, cuando se recibe la respuesta del servidor, la aplicación ejecuta de forma automática la función establecida anteriormente.

```
function muestraContenido() {
  if(peticion_http.readyState == 4) {
    if(peticion_http.status == 200) {
      alert(peticion_http.responseText);
    }
  }
}
```

La función muestraContenido() comprueba en primer lugar que se ha recibido la respuesta del servidor (mediante el valor de la propiedad readyState). Si se ha recibido alguna respuesta, se comprueba que sea válida y correcta (comprobando si el código de estado HTTP devuelto es igual a 200). Una vez realizadas las comprobaciones, simplemente se muestra por pantalla el contenido de la respuesta del servidor (en este caso, el contenido del archivo solicitado) mediante la propiedad responseText.

Refactorizando la primera aplicación

La primera aplicación AJAX mostrada anteriormente presenta algunas carencias importantes. A continuación, se refactoriza su código ampliándolo y mejorándolo para que se adapte mejor a otras situaciones. En primer lugar, se definen unas variables que se utilizan en la función que procesa la respuesta del servidor:

```
var READY_STATE_UNINITIALIZED = 0;
var READY_STATE_LOADING = 1;
var READY_STATE_LOADED = 2;
var READY_STATE_INTERACTIVE = 3;
var READY_STATE_COMPLETE = 4;
```

Como se verá más adelante, la respuesta del servidor sólo puede corresponder a alguno de los cinco estados definidos por las variables anteriores. De esta forma, el código puede utilizar el nombre de cada estado en vez de su valor numérico, por lo que se facilita la lectura y el mantenimiento de las aplicaciones.

Además, la variable que almacena la instancia del objeto XMLHttpRequest se va a transformar en una variable global, de forma que todas las funciones que hacen uso de ese objeto tengan acceso directo al mismo:

```
var peticion_http;
```

A continuación, se crea una función genérica de carga de contenidos mediante AJAX:

```
function cargaContenido(url, metodo, funcion) {
  peticion_http = inicializa_xhr();
```

```
  if(peticion_http) {
    peticion_http.onreadystatechange = funcion;
    peticion_http.open(metodo, url, true);
    peticion_http.send(null);
  }
}
```

La función definida admite tres parámetros: la URL del contenido que se va a cargar, el método utilizado para realizar la petición HTTP y una referencia a la función que procesa la respuesta del servidor.

En primer lugar, la función cargaContenido() inicializa el objeto XMLHttpRequest (llamado xhr de forma abreviada). Una vez inicializado, se emplea el objeto peticion_http para establecer la función que procesa la respuesta del servidor. Por último, la función cargaContenido() realiza la petición al servidor empleando la URL y el método HTTP indicados como parámetros.

La función inicializa_xhr() se emplea para encapsular la creación del objeto XMLHttpRequest:

```
function inicializa_xhr() {
  if(window.XMLHttpRequest) {
    return new XMLHttpRequest();
  }
  else if(window.ActiveXObject) {
    return new ActiveXObject("Microsoft.XMLHTTP");
  }
}
```

La función muestraContenido() también se refactoriza para emplear las variables globales definidas:

```
function muestraContenido() {
  if(peticion_http.readyState == READY_STATE_COMPLETE) {
    if(peticion_http.status == 200) {
      alert(peticion_http.responseText);
    }
  }
}
```

Por último, la función descargaArchivo() simplemente realiza una llamada a la función cargaContenido() con los parámetros adecuados:

```
function descargaArchivo() {
  cargaContenido("http://localhost/holamundo.txt", "GET", muestraContenido);
}
```

A continuación se muestra el código completo de la refactorización de la primera aplicación:

```
<!DOCTYPE html >
<html>
  <head>
    <title>Hola Mundo con AJAX, version 2</title>

    <script type="text/javascript" language="javascript">
```

```
      var READY_STATE_UNINITIALIZED=0;
      var READY_STATE_LOADING=1;
      var READY_STATE_LOADED=2;
var READY_STATE_INTERACTIVE=3;
var READY_STATE_COMPLETE=4;

var peticion_http;

function cargaContenido(url, metodo, funcion) {
  peticion_http = inicializa_xhr();

  if(peticion_http) {
    peticion_http.onreadystatechange = funcion;
    peticion_http.open(metodo, url, true);
    peticion_http.send(null);
  }
}

function inicializa_xhr() {
  if(window.XMLHttpRequest) {
    return new XMLHttpRequest();
  }
  else if(window.ActiveXObject) {
    return new ActiveXObject("Microsoft.XMLHTTP");
  }
}

function muestraContenido() {
  if(peticion_http.readyState == READY_STATE_COMPLETE) {
    if(peticion_http.status == 200) {
      alert(peticion_http.responseText);
    }
  }
}
function descargaArchivo() {
  cargaContenido("http://localhost/holamundo.txt", "GET", muestraContenido);
}

window.onload = descargaArchivo;

</script>

</head>
<body></body>
</html>
```

Métodos y propiedades del objeto XMLHttpRequest

El objeto `XMLHttpRequest` posee muchas otras propiedades y métodos diferentes a las manejadas por la primera aplicación de AJAX. A continuación se incluye la lista completa de todas las propiedades y métodos del objeto y todos los valores numéricos de sus propiedades.

Las propiedades definidas para el objeto `XMLHttpRequest` son:

Propiedad	Descripción
`readyState`	Valor numérico (entero) que almacena el estado de la petición
`responseText`	El contenido de la respuesta del servidor en forma de cadena de texto
`responseXML`	El contenido de la respuesta del servidor en formato XML. El objeto devuelto se puede objeto DOM
`status`	El código de estado HTTP devuelto por el servidor (`200` para una respuesta correcta, `4` encontrado", `500` para un error de servidor, etc.)
`statusText`	El código de estado HTTP devuelto por el servidor en forma de cadena de texto: "OK", "Internal Server Error", etc.

Los valores definidos para la propiedad `readyState` son los siguientes:

Valor	Descripción
`0`	No inicializado (objeto creado, pero no se ha invocado el método `open`)
`1`	Cargando (objeto creado, pero no se ha invocado el método `send`)
`2`	Cargado (se ha invocado el método `send`, pero el servidor aún no ha respondido)
`3`	Interactivo (se han recibido algunos datos, aunque no se puede emplear la propiedad `response`
`4`	Completo (se han recibido todos los datos de la respuesta del servidor)

Los métodos disponibles para el objeto `XMLHttpRequest` son los siguientes:

Método	Descripción
`abort()`	Detiene la petición actual
`getAllResponseHeaders()`	Devuelve una cadena de texto con todas las cabeceras de la r
`getResponseHeader("cabecera")`	Devuelve una cadena de texto con el contenido de la cabecer
`onreadystatechange`	Responsable de manejar los eventos que se producen. Se inv produce un cambio en el estado de la petición HTTP. Norma referencia a una función JavaScript
`open("metodo", "url")`	Establece los parámetros de la petición que se realiza al servi necesarios son el método HTTP empleado y la URL destino forma absoluta o relativa)
`send(contenido)`	Realiza la petición HTTP al servidor
`setRequestHeader("cabecera", "valor")`	Permite establecer cabeceras personalizadas en la petición H el método `open()` antes que `setRequestHeader()`

El método `open()` requiere dos parámetros (método HTTP y URL) y acepta de forma opcional otros tres parámetros. Definición formal del método `open()`:

```
open(string metodo, string URL [,boolean asincrono, string usuario, string password]);
```

Por defecto, las peticiones realizadas son asíncronas. Si se indica un valor `false` al tercer parámetro, la petición se realiza de forma síncrona, esto es, se detiene la ejecución de la aplicación hasta que se recibe de forma completa la respuesta del servidor.

No obstante, las peticiones síncronas son justamente contrarias a la filosofía de AJAX. El motivo es que una petición síncrona *congela* el navegador y no permite al usuario realizar ninguna acción hasta que no se haya recibido la respuesta completa del servidor. La sensación que provoca es que el navegador se ha *colgado* por lo que no se

recomienda el uso de peticiones síncronas salvo que sea imprescindible.

Los últimos dos parámetros opcionales permiten indicar un nombre de usuario y una contraseña válidos para acceder al recurso solicitado.

Por otra parte, el método `send()` requiere de un parámetro que indica la información que se va a enviar al servidor junto con la petición HTTP. Si no se envían datos, se debe indicar un valor igual a `null`. En otro caso, se puede indicar como parámetro una cadena de texto, un array de bytes o un objeto XML DOM.