

Ajax, la nueva tendencia en la Web

Autor: Lic. Ivannis Suárez Jérez
Profesor Instructor

Ciudad de la Habana, 4 de febrero de 2008

Resumen

El propósito de este material es aportar los fundamentos necesarios sobre la tecnología de nuevo tipo Ajax, donde a través de sencillos ejemplos un usuario con conocimientos básicos sobre informática pueda ser capaz de empezar a aprovechar las ventajas que brinda.

Palabras claves

Ajax, Javascript, aplicaciones, web, tecnología.

Índice

Resumen.....	1
Introducción	4
Definiendo Ajax.....	5
Ventajas y desventajas de Ajax.....	5
Ventajas	5
Desventajas.....	5
Navegadores que no permiten Ajax	6
El objeto XMLHttpRequest	6
Atributos	7
Onreadystatechange	7
ReadyState	7
ResponseText.....	9
ResponseXML	9
Status.....	10
StatusText.....	11
Métodos.....	12
Abort	12
GetAllResponseHeaders	13
GetResponseHeader	14
Open	15
Send	15
SetRequestHeader	15
Conclusiones	17
Bibliografía.....	17

Introducción

Desde su surgimiento las aplicaciones web han ido aumentando en calidad y servicios día a día para facilitar y optimizar la vida de los internautas. Estas proveen una forma de distribuir software que evita que los usuarios se conviertan en administradores de sistemas. Son programas que corren en servidores web y utilizan páginas web como interfaz de usuario.

En las aplicaciones web el usuario final interactúa a través de una interfaz y estas realizan tareas útiles, accediendo a un servidor Web a través de Internet o de una intranet. La habilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software en miles de potenciales clientes es una de las razones de su popularidad.

Una ventaja importante de cualquier aplicación web es que no tiene ningún costo de distribución. Los usuarios ya tienen instalada la única parte de la aplicación que necesitan en el cliente, el navegador.

Alta disponibilidad, ya que se puede realizar consultas en cualquier parte del mundo donde se tenga acceso a Internet y a cualquier hora. Se reduce el costo de mantenimiento debido a que el mantenimiento solo debe darse en el servidor y son compatibles con cualquier sistema operativo.

Ajax, concepto muy novedoso hace referencia a una combinación de tecnologías que facilitan el diseño de aplicaciones web con ciertas características avanzadas.

El uso de Ajax en la realización de aplicaciones es opcional pero brinda la posibilidad de crear aplicaciones web enriquecidas y muy interactivas con los usuarios. Asimismo admite procesar información sin considerables pausas y sin perder el estado. Permite aunar las características de las aplicaciones de escritorio (riqueza y respuesta) con la simplicidad que ha permitido la rápida proliferación de la web.

Definiendo Ajax

AJAX, proveniente del inglés **A**synchronous **J**ava**S**cript **A**nd **X**ML es un conjunto de tecnologías que nos permite crear páginas web más interactivas y parecidas cada vez más a aplicaciones de escritorios.

Combinando las tecnologías ya existentes

- ✓ XHTML (o HTML) y CSS para el diseño de las páginas web.
- ✓ JavaScript para ejecutar acciones en el cliente.
- ✓ DOM para mostrar e interactuar dinámicamente con la información presentada.
- ✓ XML para el envío y recepción de los datos entre el cliente y el servidor.
- ✓ PHP o algún otro lenguaje que se ejecute en el servidor (ASP.Net/JSP).

AJAX se destaca por permitir realizar peticiones asincrónicas al servidor en segundo plano y actualizar parte de la página sin necesidad de refrescarla completamente aumentando la interactividad, velocidad y usabilidad en la misma.

Ventajas y desventajas de Ajax

Ventajas

- ✓ Utiliza tecnologías ya existentes.
- ✓ Soportada por la mayoría de los navegadores modernos.
- ✓ Interactividad. El usuario no tiene que esperar hasta que lleguen los datos del servidor.
- ✓ Portabilidad (no requiere plug-in como Flash y Applet de Java).
- ✓ Mayor velocidad, debido a no hay que retornar toda la página nuevamente.
- ✓ La página se asemeja a una aplicación de escritorio.

Desventajas

- ✓ Se pierde el concepto de volver a la página anterior.
- ✓ Si se guarda en favoritos no necesariamente al visitar nuevamente el sitio se ubique donde nos encontrábamos al grabarla.
- ✓ La existencia de páginas con AJAX y otras sin esta tecnología hace que el usuario se desoriente.
- ✓ Problemas con navegadores antiguos que no implementan esta tecnología.
- ✓ No funciona si el usuario tiene desactivado el JavaScript en su navegador.

- ✓ Requiere programadores que conozcan todas las tecnologías que intervienen en AJAX.
- ✓ Dependiendo de la carga del servidor podemos experimentar tiempos tardíos de respuesta que desconciertan al visitante.

Navegadores que no permiten Ajax

- ✓ Opera 7 y anteriores.
- ✓ Microsoft Internet Explorer para Windows versión 4.0 y anteriores.
- ✓ Microsoft Internet Explorer para Macintosh, todas las versiones.
- ✓ Dillo.
- ✓ Navegadores basados en texto como Lynx y Links.
- ✓ Navegadores para incapacitados visuales (braille).

El objeto XMLHttpRequest

El objeto XMLHttpRequest es la vía fundamental para la comunicación entre el cliente y el servidor. Nos permite la transferencia de datos en formato XML o en otro formato entre ambos. La creación de este objeto depende del navegador que estemos usando, por ejemplo, Microsoft no lo incorpora en Javascript sino como objeto ActiveX en su navegador Internet Explorer a partir de la versión 5, en cambio para FireFox y otros navegadores lo incorpora JavaScript por lo que se procede de manera distinta.

La siguiente función Javascript nos permite retornar un objeto XMLHttpRequest haciendo transparente el navegador que estamos usando.

```
<script language="JavaScript">
function objetoXMLHttpRequest()
{
    var xmlhttp=null;
    if (window.ActiveXObject)//Para Internet Explorer.
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    else
        if (window.XMLHttpRequest)//Para Firefox y otros navegadores.
            xmlhttp = new XMLHttpRequest();
    return xmlhttp;
}
</script>
```

Atributos

Onreadystatechange

El atributo **onreadystatechange** almacena el nombre de la función que se ejecutará cuando el objeto XMLHttpRequest cambie el estado de la conexión (**readyState**).

Generalmente se usa esta función para procesar los datos recibidos desde el servidor. Para ello es necesario comprobar el estado del atributo **readyState**, ya que solo se tendrá acceso a la respuesta enviada desde el servidor en los atributos **responseXML** y **responseText** cuando el **readyState** tenga valor de 3 ó 4.

Ejemplo

```
<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
    // Muestra el estado de la petición.
    alert ( objXMLHttpRequest.readyState );
}
//Crea el objeto.
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
//Otros métodos que se verán más adelante.
objXMLHttpRequest.open("GET", "tupagina.html", true);
objXMLHttpRequest.send(null);
</script>
```

ReadyState

EL atributo **readyState** devuelve el estado actual de la petición hecha a través del objeto XMLHttpRequest al servidor. Cada vez que cambia el valor de **readyState** se ejecuta la función definida en el atributo **onreadystatechange**.

Estados en los que puede estar el atributo:

- ✓ 0 No inicializado (el método open no ha sido llamado).
- ✓ 1 Cargando (se llamó al método open).
- ✓ 2 Cargado (se llamó al método send y ya tenemos la cabecera de la petición HTTP y el status).
- ✓ 3 Interactivo (la propiedad responseText tiene datos parciales).

- ✓ 4 Completado (la propiedad responseText tiene todos los datos pedidos al servidor).

Ejemplo

```
<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
    // Estado de la petición.
    iEstado = objXMLHttpRequest.readyState;
    switch(iEstado)
    {
        case 0 : // 0 No inicializado (el método open no ha sido llamado).
            alert("No inicializado (el método open no ha sido llamado)");
            break;
        case 1: // 1 Cargando (se llamó al método open).
            alert("Cargando (se llamó al método open)");
            break;
        case 2: // 2 Cargado (se llamó al método send y ya tenemos la cabecera
de la petición HTTP y el status).
            alert("Cargado (se llamó al método send y ya tenemos la
cabecera de la petición HTTP y el status)");
            break;
        case 3: // 3 Interactivo (la propiedad responseText tiene datos
parciales).
            alert("Interactivo (la propiedad responseText tiene datos
parciales)");
            break;
        case 4: // 4 Completado (la propiedad responseText tiene todos los
datos pedidos al servidor).
            alert("Completado (la propiedad responseText tiene todos los
datos pedidos al servidor)")
            break;
        default:
            break;
    }
}
//Crea el objeto.
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
//Otros métodos que se verán más adelante.
objXMLHttpRequest.open("GET", "tupagina.html", true);
objXMLHttpRequest.send(null);
</script>
```

ResponseText

El atributo **responseText** devuelve los datos recibidos desde el servidor en formato de cadena.

Ejemplo

```
<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
    // Muestra el estado de la petición.
    switch(objXMLHttpRequest.readyState)
    {
        case 2: // 2 Cargado (se llamó al método send y ya tenemos la cabecera
de la petición HTTP y el status).
            alert("Cargando...");
            break;
        case 4: // 4 Completado (la propiedad responseText tiene todos los
datos pedidos al servidor).
            iDatos = objXMLHttpRequest.responseText;
            alert(iDatos);
            break;
        default:
            break;
    }
}
//Crea el objeto.
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
//Otros métodos que se verán más adelante.
objXMLHttpRequest.open("GET", "tupagina.html", true);
objXMLHttpRequest.send(null);
</script>
```

ResponseXML

El atributo **responseXML** devuelve los datos recibidos desde el servidor en forma de documento XML.

Esta propiedad retorna *null* en caso de que la respuesta del servidor no tenga el encabezado *text/xml*, *application/xml* o acabe en *+xml*. Para tratar el documento XML recibido es necesario utilizar las propiedades del DOM.

Ejemplo

```

<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
    // Muestra el estado de la petición.
switch(objXMLHttpRequest.readyState)
{
    case 2: // 2 Cargado (se llamó al método send y ya tenemos la cabecera
de la petición HTTP y el status).
        alert("Cargando...");
        break;
    case 4: // 4 Completado (la propiedad responseText tiene todos los
datos pedidos al servidor).
        // Acceder al documento XML.
        iDocumento = objXMLHttpRequest.responseXML.documentElement;
        // Mostrar la cantidad de item's.
        if (iDocumento <> null)
            alert
                (
oDocumento.getElementsByTagName('item').length );
        break;
    default:
        break;
}
}
//Crea el objeto.
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
//Otros métodos que se verán más adelante.
objXMLHttpRequest.open("GET", "tupagina.html", true);
objXMLHttpRequest.send(null);
</script>

```

Status

El atributo **status** devuelve un código enviado por el servidor del estado HTTP del envío. Esta propiedad se utiliza para comprobar el estado de la comunicación con el servidor, siendo correcta cuando el atributo sea igual a 200.

Estados en los que puede estar este atributo:

- ✓ 100 Continua
- ✓ 101 Cambio de protocolo
- ✓ 200 OK
- ✓ 201 Creado

- ✓ Entre otros estados

Ejemplo

```
<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
    // Muestra el estado de la petición.
    switch(objXMLHttpRequest.readyState)
    {
        case 2: // 2 Cargado (se llamó al método send y ya tenemos la cabecera
de la petición HTTP y el status).
            alert("Cargando...");
            break;
        case 4: // 4 Completado (la propiedad responseText tiene todos los
datos pedidos al servidor).
            //Chequeando el estado.
            if ( objXMLHttpRequest.status == 200 )
            {
                iDatos = objXMLHttpRequest.responseText;
                alert (iDatos);
            }
            else
                alert("Ha ocurrido un error el servidor")
            break;
        default:
            break;
    }
}
//Crea el objeto.
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
//Otros métodos que se verán más adelante.
objXMLHttpRequest.open("GET", "tupagina.html", true);
objXMLHttpRequest.send(null);
</script>
```

StatusText

El atributo **statusText** devuelve el texto con el estado HTTP del envío al servidor. Esta propiedad se utiliza para comprobar o mostrar el estado de la comunicación con el servidor, siendo correcta cuando el atributo sea igual a "OK".

Métodos

El objeto XMLHttpRequest posee un conjunto de métodos que nos serán de gran ayuda para el tratamiento de las peticiones asincrónicas al servidor.

Abort

El método **abort** tiene por objetivo detener las peticiones o conexiones establecidas con el servidor, poniendo a cero el valor del atributo **readyState**.

Es muy útil en situaciones en las que el servidor se encuentra saturado y no puede responder a una petición, en este caso haciendo una llamada al método **abort** podemos cancelar dicha petición. Una técnica usual puede ser fijar un tiempo determinado para completar la petición, y si en ese tiempo el servidor no ha contestado, cancelar mediante el método **abort** y mostrar un mensaje al usuario.

Ejemplo

En este ejemplo se hace una llamada al método **abort** después de haber enviado una petición al servidor. Como resultado no se debe mostrar nada, ya que se canceló la petición enviada.

```
<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
    // Muestra el estado de la petición.
    switch(objXMLHttpRequest.readyState)
    {
        case 2: // 2 Cargado (se llamó al método send y ya tenemos la cabecera
de la petición HTTP y el status).
            alert("Cargando...");
            break;
        case 4: // 4 Completado (la propiedad responseText tiene todos los
datos pedidos al servidor).
            //Chequeando el estado.
            if ( objXMLHttpRequest.status == 200 )
            {
                iDatos = objXMLHttpRequest.responseText;
                alert (iDatos);
            }
            else
                alert(objXMLHttpRequest.statusText)
            break;
        default:
```

```

        break;
    }
}
//Crea el objeto.
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
//Otros métodos que se verán más adelante.
objXMLHttpRequest.open("GET", "tupagina.html", true);
objXMLHttpRequest.send(null);
//Cancelamos la petición.
objXMLHttpRequest.abort();
</script>

```

GetAllResponseHeaders

El método **getAllResponseHeaders** devuelve todas las cabeceras HTTP en forma de cadena, recibidas de una petición enviada al servidor.

Ejemplo

```

<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
    // Muestra el estado de la petición.
    switch(objXMLHttpRequest.readyState)
    {
        case 2: // 2 Cargado (se llamó al método send y ya tenemos la cabecera
de la petición HTTP y el status).
            alert("Cargando...");
            break;
        case 4: // 4 Completado (la propiedad responseText tiene todos los
datos pedidos al servidor).
            //Chequeando el estado.
            if ( objXMLHttpRequest.status == 200 )
            {
                iHeaders = objXMLHttpRequest.getAllResponseHeaders();
                alert(iHeaders)
            }
            else
                alert(objXMLHttpRequest.statusText)

            break;
        default:
            break;
    }
}

```

```

}
//Crea el objeto.
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
//Otros métodos que se verán más adelante.
objXMLHttpRequest.open("GET", "tupagina.html", true);
objXMLHttpRequest.send(null);
</script>

```

GetResponseHeader

El método **getResponseHeader(header)** devuelve el valor de una cabecera HTTP pasada por parámetro.

Header – Nombre de la cabecera HTTP que se quiere conocer.

Ejemplo

```

<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
// Muestra el estado de la petición.
switch(objXMLHttpRequest.readyState)
{
case 2: // 2 Cargado (se llamó al método send y ya tenemos la cabecera
de la petición HTTP y el status).
alert("Cargando...");
break;
case 4: // 4 Completado (la propiedad responseText tiene todos los
datos pedidos al servidor).
//Chequeando el estado.
if ( objXMLHttpRequest.status == 200 )
{
iHeaders =
objXMLHttpRequest.getResponseHeader("Content-Type");
alert(iHeaders)
}
else
alert(objXMLHttpRequest.statusText)
break;
default:
break;
}
}
//Crea el objeto.

```

```
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
//Otros métodos que se verán más adelante.
objXMLHttpRequest.open("GET", "tupagina.html", true);
objXMLHttpRequest.send(null);
</script>
```

Open

El método **open**(método, URL [, asíncrona [, nombre [, password]]]) abre y prepara el objeto para una nueva conexión.

Método – Método de conexión con el servidor (GET o POST). Se utiliza el método GET cuando deseamos que el navegador guarde datos en cache y POST para obligar a descargar todos los datos en cada petición.

URL – Url de destino.

Asíncrona – Parámetro opcional para usar el modo asíncrono (true) o síncrono (false).

Nombre, password – Parámetros opcionales solo aplicables para la autenticación en el servidor.

Send

El método **send**(contenido) permite enviar todo el contenido pasado por parámetro al servidor.

Contenido – Puede ser una cadena de texto o una referencia al DOM. Se aconseja siempre pasar este parámetro para evitar problemas con algunos navegadores.

SetRequestHeader

El método **setRequestHeader**(nombre, valor) adiciona una cabecera HTTP a la petición que se va a realizar al servidor.

Nombre – Nombre de la cabecera.

Valor – Valor de la cabecera.

Ejemplo

```
<script language="JavaScript">
// Función que se ejecutará.
function pFuncion () {
// Muestra el estado de la petición.
switch(objXMLHttpRequest.readyState)
{
```

```

        case 2: // 2 Cargado (se llamó al método send y ya tenemos la cabecera
de la petición HTTP y el status).
            alert("Cargando...");
            break;
        case 4: // 4 Completado (la propiedad.responseText tiene todos los
datos pedidos al servidor).
            //Chequeando el estado.
            if ( objXMLHttpRequest.status == 200 )
            {
                iDatos = objXMLHttpRequest.responseText;
                alert (iDatos);
            }
            else
                alert(objXMLHttpRequest.statusText)
            break;
        default:
            break;
    }
}
//Crea el objeto.
objXMLHttpRequest = new objetoXMLHttpRequest();
// Se define la función que se ejecutará.
objXMLHttpRequest.onreadystatechange = pFuncion;
objXMLHttpRequest.setRequestHeader ("Accept-Language", "sp");
objXMLHttpRequest.open("POST", "tupagina.html", true);
objXMLHttpRequest.send(null);
</script>

```

Conclusiones

La evolución de Ajax ha sido un hecho que ha marcado la tendencia en los lenguajes de programación en estos últimos años. Por muchos desarrolladores es sabido que Ajax nos lleva a hacer aplicaciones más rápidas, eficientes y dinámicas.

No decir al usuario que está sucediendo es uno de los errores más repetidos, pues deben adaptarse a la nueva forma de pensar que requiere desarrollar en Ajax y darse cuenta de que la carga de las páginas no se hace de la manera convencional y el navegador por lo tanto no la interpreta como tal.

Ajax sin lugar a duda es una nueva alternativa para garantizar la calidad de las aplicaciones web. Los mayores desafíos al crear aplicaciones Ajax no son técnicas. Las tecnologías centrales son maduras, estables y bien conocidas. En cambio, los desafíos son para los diseñadores de estas aplicaciones: olvidar que creen saber sobre las limitaciones de la web, y comenzar a imaginar un rango más amplio y rico de posibilidades.

Bibliografía

- ✓ www.ajaxya.com.ar
- ✓ www.programacionweb.net
- ✓ www.wikipedia.com