

# DESARROLLO WEB EN ENTORNO CLIENTE

## CAPÍTULO 8:

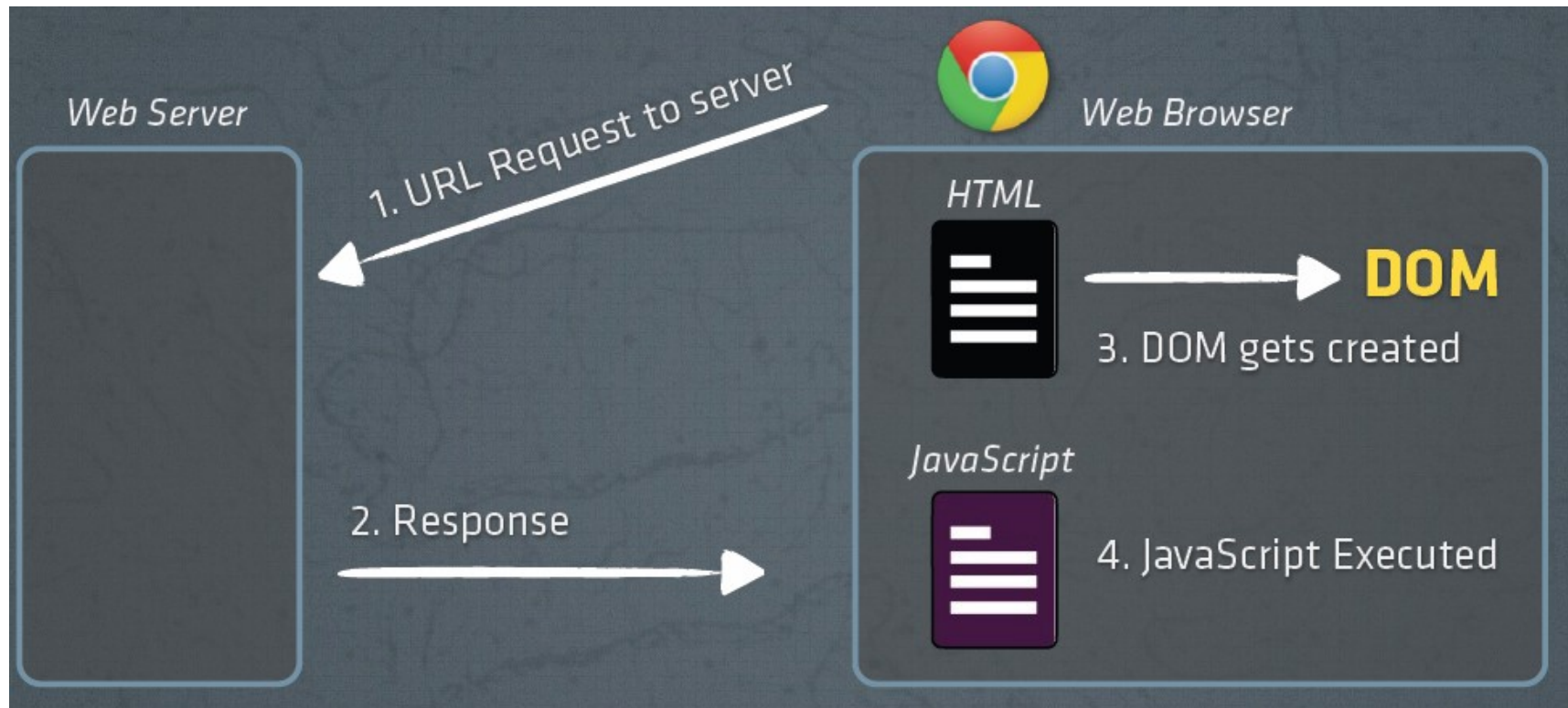
Utilización de Mecanismos de Comunicación Asíncrona

# Objetivos

- 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.

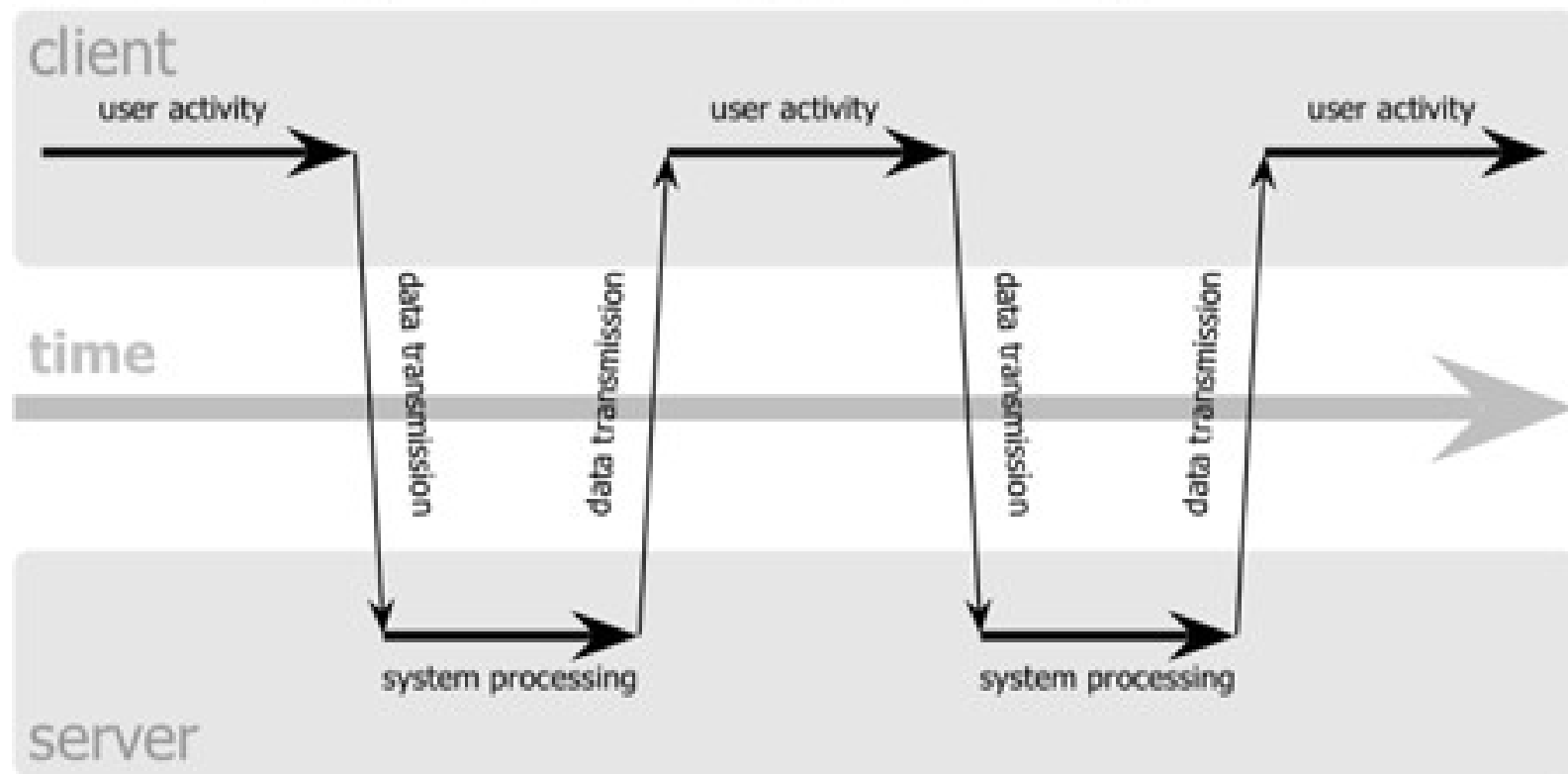
# Mecanismos de comunicación síncrona

- 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 síncrona

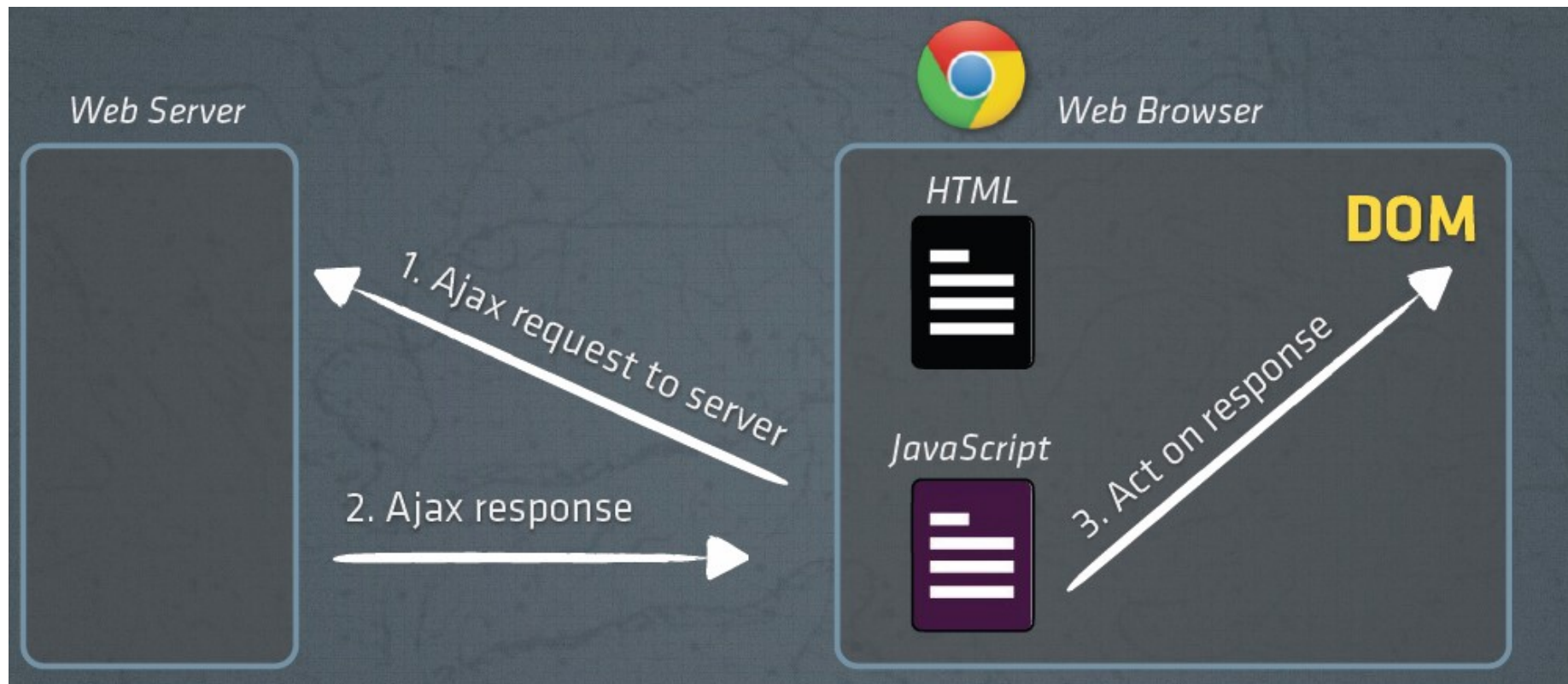
classic web application model (synchronous)



# Mecanismos de comunicación asíncrona

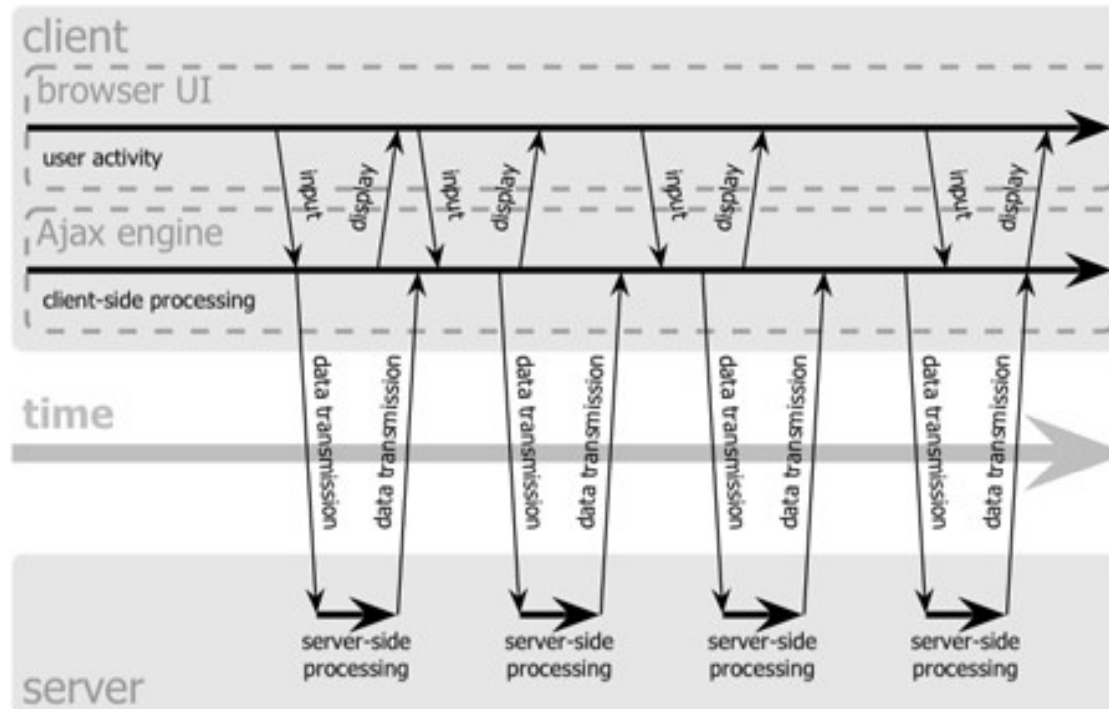
- El mecanismo de comunicación **asíncrona** recarga en **segundo plano** una **parte** de la página Web, dejando **desbloqueado** el resto.
- El cliente que envía una petición no permanece bloqueado esperando la respuesta del servidor.
- Esto ayuda a que las aplicaciones Web tengan una interactividad similar a las aplicaciones de escritorio.
- Es en parte lo que hace algunos años se denomina Web 2.0.

# Mecanismos de comunicación asíncrona

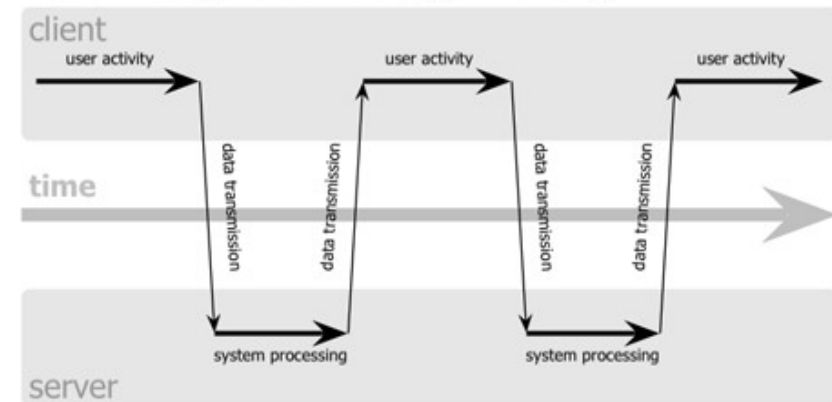


# Mecanismos de comunicación asíncrona

## Ajax web application model (asynchronous)



## classic web application model (synchronous)



# AJAX

- Necesidad => aplicaciones Web interactivas.
- Solución => nuevo uso a tecnologías como XML, CSS o DOM.
- En 2005 J.J. Garrett habla por primera vez sobre AJAX
- ¿Qué es AJAX?

Acrónimo de *Asynchronous JavaScript And XML*

- No es ninguna tecnología, ni lenguaje de programación
- Es una técnica de desarrollo web que combina varias tecnologías consiguiendo una navegación más ágil y rápida, más dinámica.
- Se suprimen los efectos secundarios de las recargas, como la pérdida del contexto, la ubicación del *scroll* o las respuestas más lentas.



# Tecnologías en AJAX

- **XHTML ó HTML 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.

# Elección de AJAX

- Uso de nuevas tecnologías y mayor complejidad.
- El comportamiento considerado lógico por el usuario al utilizar la funcionalidad de “volver a la página anterior” no es reproducido de la misma manera.
- 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.

# Elección de AJAX

- **¿Dónde utilizar AJAX?**
  - Comunicación rápida entre usuarios
  - Interacción a través de formularios
  - Votaciones, encuestas, valoraciones, etc.
  - Filtrado y manipulación de datos o resultados de búsqueda
  - Autocompletado de campos de texto usados comúnmente
  - Ejemplos:
    - Mini álbum de fotos, buscador sencillo, sugerir valores para un campo, paginar resultados, editar campos no editables y actualizar contenido,...

# Elección de AJAX

- **¿Dónde no utilizar AJAX?**
  - Envíos a través de formularios simples
  - Búsquedas
  - Navegación básica
  - Reemplazar grandes cantidades de texto
  - Manipulación de la interfaz
  - Validación de usuarios

# Elección de AJAX

## • Ventajas e inconvenientes

VENTAJAS	INCOVENIENTES
<ul style="list-style-type: none"><li>• Mejor experiencia de usuario<ul style="list-style-type: none"><li>• Recuperación asíncrona</li><li>• Interfaz de escritorio en la web</li></ul></li><li>• Menos ancho de banda</li><li>• Menos proceso en el servidor</li><li>• No precisa plugins</li></ul>	<ul style="list-style-type: none"><li>• Problemas de accesibilidad, compatibilidad, seguridad,...</li><li>• Pérdida de funcionalidades del navegador:<ul style="list-style-type: none"><li>• Historial</li><li>• Favoritos o bookmarks</li></ul></li><li>• Peor indexación en los motores de búsqueda</li><li>• Demasiado código AJAX hace lento el navegador</li></ul>

# El objeto XMLHttpRequest

- Aparece a partir de Internet Explorer 5 en la forma de un control ActiveX llamado XMLHttpRequest.
- Se fueron transformando en un estándar de facto en navegadores como Firefox, Safari y Opera.
- 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.

# El objeto XmlHttpRequest

- Comunicación GET/POST
- Documentos pueden ser texto plano/xml
- Trabaja en background
- Número limitado de peticiones
- Permite especificar un manejador para el control de cambios de estado
- Manejador notifica el estado de la petición:
  - Inicializada
  - Iniciada
  - En proceso de retornar la información
  - Operación completada

# Atributos del objeto XMLHttpRequest

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. <b>404</b> para "Not Found").
statusText	Devuelve el estado como una cadena (p. ej. " <b>Not Found</b> ").



# Métodos del objeto XMLHttpRequest

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])</code>	<p>Especifica el método, URL y otros atributos opcionales de una petición.</p> <p>El parámetro de <b>método</b> puede tomar los valores "GET" o "POST".</p> <p>El parámetro <b>URL</b> puede ser una URL relativa o absoluta.</p> <p>El parámetro <b>asíncrono</b> especifica si la petición será gestionada asíncronamente o no. El valor por defecto es <b>true</b></p>

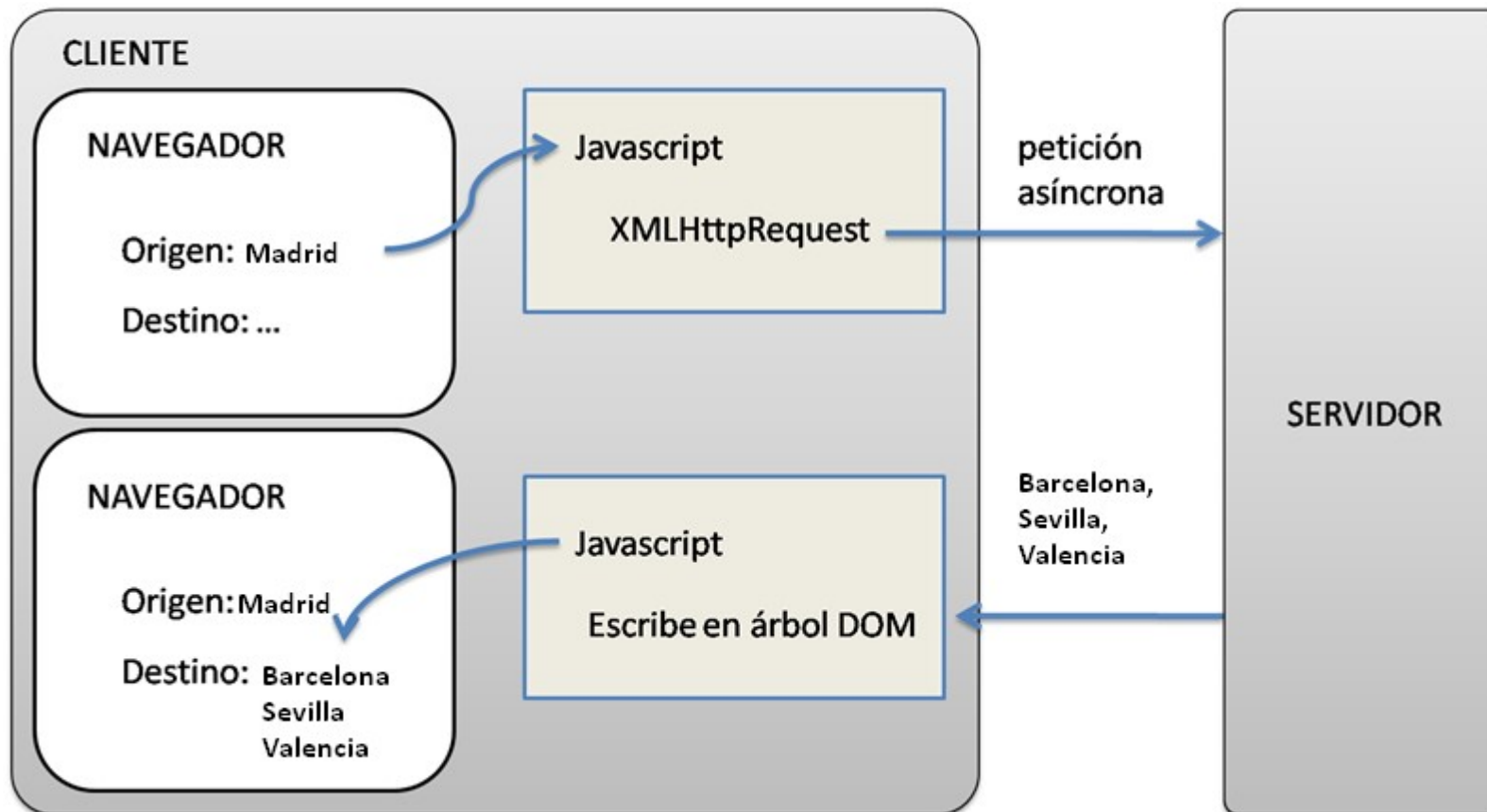
# Métodos del objeto XMLHttpRequest

Métodos	Descripción
<code>setRequestHeader(cabecera, valor)</code>	Añade la cabecera HTTP para especificar el tipo de petición que se envía a través del método <code>send()</code> .
<code>send([datos])</code>	Envía la petición

# Propiedades del objeto XMLHttpRequest

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

# Perspectiva Global con AJAX



# Interacción con el servidor: Solicitudes GET

## Ejemplos:

- Una simple petición GET:

```
XMLHttpRequest.open("GET", "demo_get.asp", true);  
XMLHttpRequest.send();
```

- En el ejemplo anterior, se puede conseguir un resultado almacenado en caché. Para evitar esto, añade un campo único a la URL:

```
XMLHttpRequest.open("GET", "demo_get.asp?t=" + Math.random(), true);  
XMLHttpRequest.send();
```

- Si desea enviar información con el método GET, agregar la información a la dirección URL:

```
XMLHttpRequest.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);  
XMLHttpRequest.send();
```

# Interacción con el servidor: Solicitudes POST

## Ejemplos:

- Una petición POST simple:

```
XMLHttpRequest.open("POST","demo_post.asp",true);  
XMLHttpRequest.send();
```

- Para publicar datos como un formulario HTML, añadir una cabecera HTTP con `setRequestHeader ()`. Especifique los datos que desea enviar en el método `send ()`:

```
XMLHttpRequest.open("POST","ajax_test.asp",true);  
XMLHttpRequest.setRequestHeader("Content-type","application/x-www-form-urlencoded");  
XMLHttpRequest.send("fname=Henry&lname=Ford");
```

# Ejemplo de AJAX

- 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.

```
<html>
<head>
  <meta charset='utf-8'>
</head>
<body>
  <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>
</body>
</html>
```

# Ejemplo de AJAX


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

```
<script language = "javascript">

    var objetoXHR = new XMLHttpRequest();

    1 function obtenerDatosServidor(urlServidor, elemento)
    {
    2     objeto_destino = document.getElementById(elemento);
    3     objetoXHR.open("GET", urlServidor, true);
    4     objetoXHR.onreadystatechange = respuesta;
    5     objetoXHR.send();
    }

    function respuesta() {
        if (objetoXHR.readyState == 4 && objetoXHR.status == 200) {
            objeto_destino.innerHTML = objetoXHR.responseText;
        }
    }
</script>
```





# Ejemplos de webs que utilizan de AJAX

- Sugerir al usuario lo que está buscando → [Kayak](#)
- Modificar el contenido de la página sin recargar → Google Calendar, Maps, ..
- Simuladores de código y compiladores online → w3schools, [http://www.tutorialspoint.com/compile\\_vb.net\\_online.php](http://www.tutorialspoint.com/compile_vb.net_online.php)

# JSON

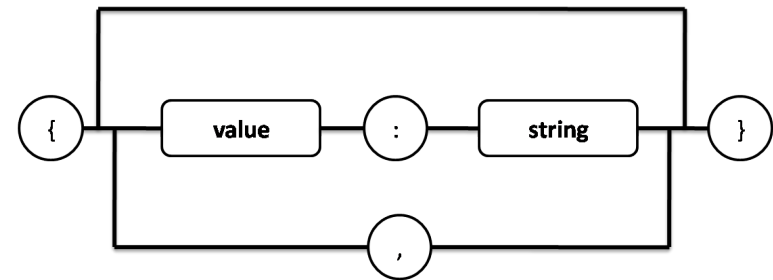
- JSON (JavaScript Object Notation).
- Formato ligero para el intercambio de datos entre distintas tecnologías.
- Es una manera de almacenar información.
- Fue pensado en un primer momento como una alternativa a XML. Éste lenguaje tienen una gran cantidad de información extra, asociada a su estructura.
- Puede ser leído por cualquier lenguaje de programación.

# JSON

- JSON está constituido por dos estructuras:
  - Una colección de pares de nombre/valor.
  - Una lista ordenada de valores.
- Ficheros JSON
  - Extensión '.json'
  - Tipo MIME : 'application/json'

# JSON: Object

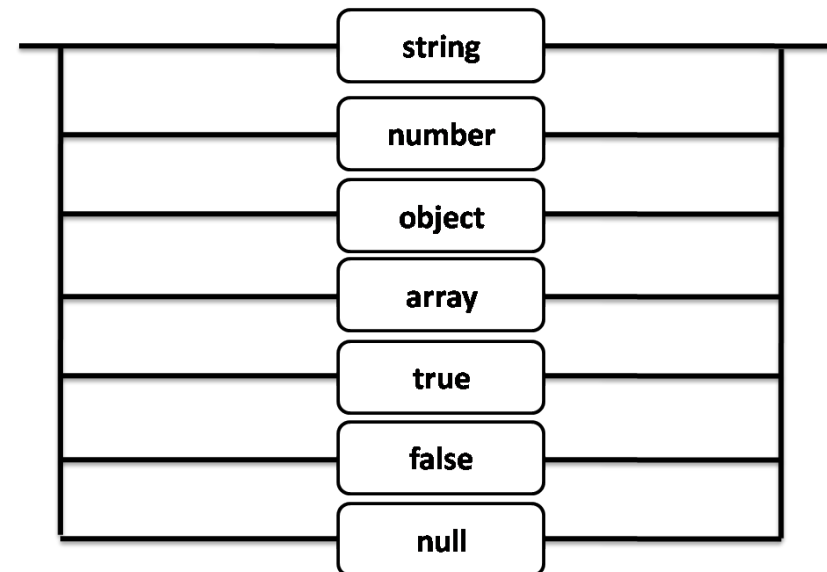
- El elemento base de la sintaxis es el *object*.
- Está conformado por un conjunto desordenado de pares nombre/valor.
- Un objeto comienza con una llave de apertura y finaliza con una llave de cierre.
- Cada nombre es seguido por dos puntos, estando los pares nombre/valor separados por una coma.



```
{ "NombreFruta" : "Manzana" , "Cantidad" : 20 }
```

# JSON: Array

- 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.



```
{  
  "Frutas": [  
    { "NombreFruta": "Manzana" , "cantidad": 10 },  
    { "NombreFruta": "Pera" , "cantidad": 20 },  
    { "NombreFruta": "Naranja" , "cantidad": 30 }  
  ]  
}
```

# JSON

- Con JavaScript se puede crear un array de objetos y asignar los datos de esta manera:

- Ejemplo

```
var employees = [  
  { "firstName": "John" , "lastName": "Doe" },  
  { "firstName": "Anna" , "lastName": "Smith" },  
  { "firstName": "Peter" , "lastName": "Jones" }  
];
```

- La primera entrada en la matriz de objetos de JavaScript puede acceder de esta manera:

```
employees[0].firstName + " " + employees[0].lastName;
```

- El contenido devuelto será:

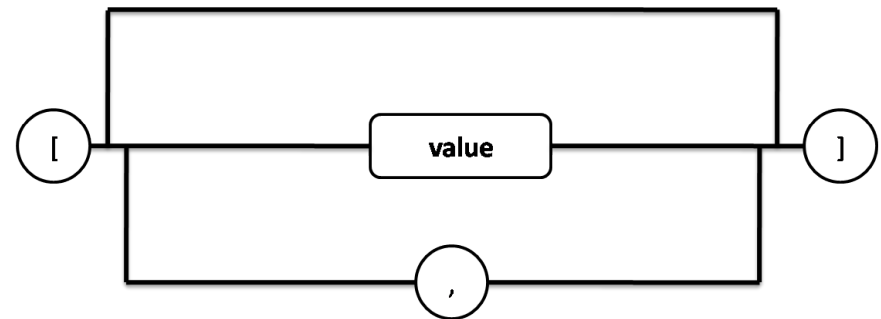
*John Doe*

- Los datos pueden ser modificados de esta manera:

```
employees[0].firstName = "Gilbert";
```

# JSON: Value

- A su vez un elemento *value* puede ser:
  - Una *string*
  - Un *number* (entero o *float*)
  - Un *booleano* (*true* o *false*)
  - Un *object* {}
  - Un *array* []
  - *null*
- Estas estructuras pueden anidarse.



# JSON: Value

- **Fruta:**
  - 10 manzanas
  - 20 Peras
  - 30 Naranjas
- **Verduras**
  - 80 lechugas
  - 15 tomates
  - 50 pepinos

```
{
  "Fruteria": [
    {
      "Fruta": [
        {
          "Nombre": "Manzana", "Cantidad": 10
        },
        {
          "Nombre": "Pera", "Cantidad": 20
        },
        {
          "Nombre": "Naranja", "Cantidad": 30
        }
      ]
    },
    {
      "Verdura": [
        {
          "Nombre": "Lechuga", "Cantidad": 80
        },
        {
          "Nombre": "Tomate", "Cantidad": 15
        },
        {
          "Nombre": "Pepino", "Cantidad": 50
        }
      ]
    }
  ]
}
```



# Ejemplo JSON y XML

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

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

# Ejemplo JSON y XML

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

```
{
  "menu": {
    "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
        {"value": "New", "onclick": "CreateNewDoc()"},
        {"value": "Open", "onclick": "OpenDoc()"},
        {"value": "Close", "onclick": "CloseDoc()"}
      ]
    }
  }
}
```

# Conversión de un texto JSON a JavaScript

- Uno de los usos más comunes es obtener datos JSON desde un servidor web (como un archivo o como HttpRequest), convertirlos en un objeto JavaScript, y luego utilizarlos en una página web.
- Crear una cadena de JavaScript que contiene la sintaxis de JSON:

```
var txt = '{ "employees" : [' +  
            '{ "firstName":"John" , "lastName":"Doe" },' +  
            '{ "firstName":"Anna" , "lastName":"Smith" },' +  
            '{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

- La función **JSON.parse(txt)** de JavaScript se puede utilizar para convertir un texto JSON en un objeto de JavaScript.
- Esta función utiliza el compilador de JavaScript que analiza el texto JSON y produce un objeto de JavaScript.

```
var obj = JSON.parse(txt);
```

# Conversión de un texto JSON a JavaScript

- Realizado este paso, ya se puede utilizar el objeto JavaScript en la página:

```
'{ "employees" : ['+  
    '{ "firstName":"John" , "lastName":"Doe" }, ' +  
    '{ "firstName":"Anna" , "lastName":"Smith" }, ' +  
    '{ "firstName":"Peter" , "lastName":"Jones" } ]} ';
```

<p>

First Name: <span id="fname"></span><br />

Last Name: <span id="lname"></span><br />

</p>

<script>

...

var obj = JSON.parse(petición.responseText);

document.getElementById("fname").innerHTML = obj.employees[1].firstName;

document.getElementById("lname").innerHTML = obj.employees[1].lastName;

</script>

# Conversión de un texto JSON a JavaScript

- Para versiones antiguas de navegadores que no tienen soporte para `JSON.parse()` se puede utilizar la función **eval ()** de JavaScript.
- Convierte un texto JSON en un objeto de JavaScript.
- El texto debe ser envuelto en paréntesis para evitar un error de sintaxis:

```
var obj = eval ("(" + txt + ")");
```

- Esta función debe ser evitada por razones de seguridad

# Librerías de Actualización Dinámica

- Junto con la tecnología AJAX están las librerías que implementan una gran cantidad de funciones y controles a ser utilizados por los desarrolladores:
  - 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.

# ANEXOS

# Errores AJAX

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	



# PHP - JSON

- **Json-encode(value)**

```
$myarray = array('Guitar' => 'Johnny', 'Vocals'=> 'Stephen',  
                'Bass' => 'Andy', 'Drums' => 'Mike');  
$myJson = json_encode($myarray);  
echo $myJson;
```

```
{"Guitar":"Johnny","Vocals":"Stephen","Bass":"Andy","Drums":"Mike"}
```

- **Json-decode(value, boolean)**

```
$myJson = '{"Guitar" : "Johnny", "Vocals": "Stephen",  
          "Bass" : "Andy", "Drums" : "Mike"}';  
$myarray = json_decode($myJson, true);  
print_r($myarray);
```

```
Array ( [Guitar] => Johnny [Vocals] => Stephen [Bass] => Andy [Drums] => Mike )
```

```
stdClass Object ( [Guitar] => Johnny [Vocals] => Stephen [Bass] => Andy [Drums] => Mike )
```

# Tipos de cabeceras – cliente (js)

## Envío de parámetros al servidor (POST)

- Parámetros clave=valor

```
peticion_http.setRequestHeader("Content-Type",  
                                "application/x-www-form-urlencoded");
```

- Parámetros en formato XML

```
peticion_http.setRequestHeader("Content-Type", "application/xml");
```

- Parámetros en formato JSON

```
http_request.setRequestHeader("Content-Type", "application/json");
```

# Tipos de cabeceras – servidor (php)

- Resultados en formato XML

```
header('Content-Type: text/xml; charset=utf-8');
```

- Resultados en formato JSON

```
header('Content-type: application/json; charset=utf-8');
```

# \$.ajax(), \$.post(), \$.get

- A partir de la versión 3.0 de jQuery han sido sustituidos algunos métodos:
  - **Success** → done
  - **Error** → fail
  - **Complete** → always

# Ejemplo \$.ajax()

```
$.ajax({
    // En data puedes utilizar un objeto JSON, un array o un query string
    data: {"parametro1" : "valor1", "parametro2" : "valor2"},
    //Cambiar a type: POST si necesario
    type: "GET",
    // Formato de datos que se espera en la respuesta
    dataType: "json",
    // URL a la que se enviará la solicitud Ajax
    url: "script.php",
})
.done(function( data, textStatus, jqXHR ) {
    if ( console && console.log ) {
        console.log( "La solicitud se ha completado correctamente." );
    }
})
.fail(function( jqXHR, textStatus, errorThrown ) {
    if ( console && console.log ) {
        console.log( "La solicitud a fallado: " + textStatus);
    }
});
```

# Ejemplo \$.getJSON()

```
$.getJSON( "script.php", { "parametro1" : "valor1", "parametro2" : "valor2" } )  
  .done(function( data, textStatus, jqXHR ) {  
    if ( console && console.log ) {  
      console.log( "La solicitud se ha completado correctamente." );  
    }  
  })  
  .fail(function( jqXHR, textStatus, errorThrown ) {  
    if ( console && console.log ) {  
      console.log( "Algo ha fallado: " + textStatus );  
    }  
  });
```

# Ejemplo \$.post()

```
$.post( "script.php", { "parametro1" : "valor1", "parametro2" : "valor2" }, null, "json" )  
  .done(function( data, textStatus, jqXHR ) {  
    if ( console && console.log ) {  
      console.log( "La solicitud se ha completado correctamente." );  
    }  
  })  
  .fail(function( jqXHR, textStatus, errorThrown ) {  
    if ( console && console.log ) {  
      console.log( "La solicitud a fallado: " +  textStatus);  
    }  
  });
```