

Programación Web Avanzada: AJAX y Google Maps

**Universidad de Colima
México**



Soporte de AJAX en PHP


Sergio Luján Mora



**Departamento de Lenguajes y
Sistemas Informáticos**

Programación Web Avanzada: AJAX y Google Maps

Universitat d'Alacant
Universidad de Alicante




Índice

- Introducción
- JSON
- XML
- ¿JSON o XML?

Programación Web Avanzada: AJAX y Google Maps

Universitat d'Alacant
Universidad de Alicante



Introducción

- PHP dispone de algunas librerías para trabajar con los formatos que emplea AJAX

Introducción

- Cuando se invoca mediante AJAX a un script en el servidor, pueden existir problemas con la caché: el navegador, un proxy o algún otro elemento intermedio pueden devolver el resultado de una petición anterior

Introducción

- Solución en el cliente:
 - Añadir un parámetro ficticio (una fecha o un número aleatorio) que cambie con cada petición para que se interprete como una nueva petición:

```
d = new Date();  
peticion.open('GET', 'http://www.ua.es/ajax.php?n=' +  
    d.getTime(), true);
```

Introducción

- Solución desde el servidor:
 - Enviar encabezados HTTP para evitar que la respuesta de PHP se almacene en la caché:

```
<?php
header("Cache-Control: no-cache, must-
revalidate");
header("Expires: Mon, 01 Jan 2007 01:00:00
GMT");
?>
```

JSON

- JSON (*JavaScript Object Notation*) es un formato de intercambio de datos ligero basado en texto
- JSON Specification RFC 4627:
 - <http://tools.ietf.org/html/rfc4627>
- JSON Validator:
 - <http://www.jsonlint.com/>

JSON

- Se emplea para la serialización de datos estructurados:
 - Arrays:
 - La lista de valores encerrados entre corchetes y separados por comas
 - Objetos:
 - Los pares propiedad/valor encerrados entre llaves y separados por comas
 - La propiedad se separa del valor por dos puntos
 - En ambos casos, las propiedades y valores de tipo cadena encerrados entre comillas dobles

JSON

- Ejemplo:

```
[1, 2, 3, 4, 5]
```

```
["Alicante", "Castellón", "Valencia"]
```

```
{"nombre": "Sergio", "apellidos": "Luján  
Mora"}
```

```
{"posicion": {"x": 10, "y": 20}}
```

JSON

- Ejemplo:

```
{  
  "Image": {  
    "Width": 800,  
    "Height": 600,  
    "Title": "View from 15th Floor",  
    "Thumbnail": {  
      "Url": "http://www.ex.com/image/481989943",  
      "Height": 125,  
      "Width": "100"  
    }  
  }  
}
```

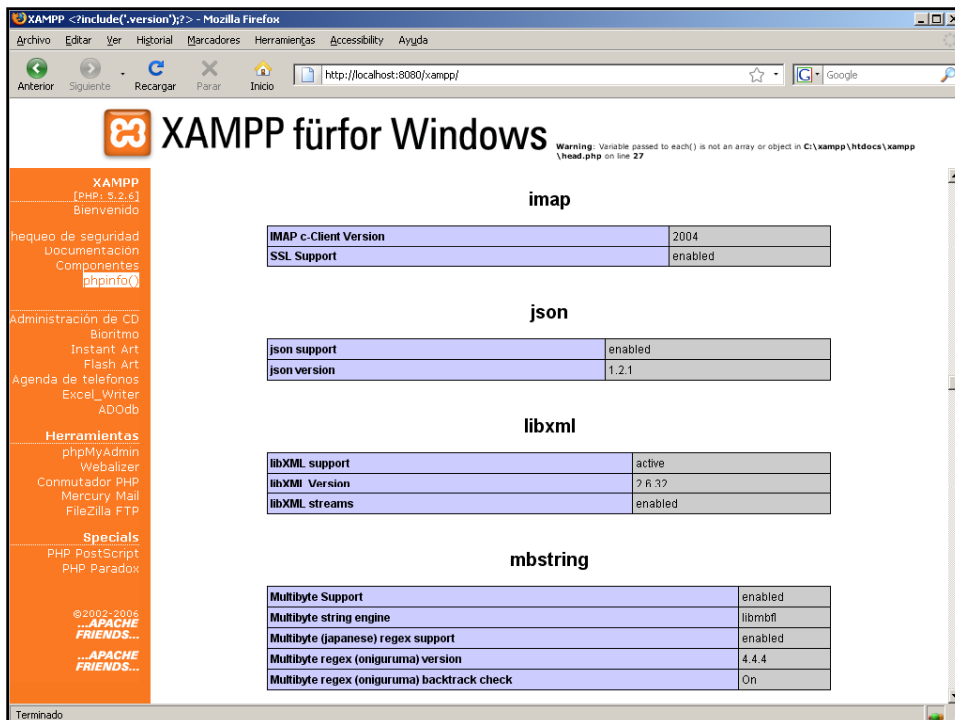
JSON

- Ejemplo:

```
[  
  { "precision": "zip", "Latitude": 37.7668,  
    "Longitude": -122.3959, "Address": "",  
    "City": "SAN FRANCISCO", "State": "CA",  
    "Zip": "94107", "Country": "US" },  
  { "precision": "zip", "Latitude": 37.371991,  
    "Longitude": -122.026020, "Address": "",  
    "City": "SUNNYVALE", "State": "CA", "Zip":  
    "94085", "Country": "US" }  
]
```

JSON

- A partir de la versión PHP 5.2.0, se incluye soporte para JSON
- Comprobar con `phpinfo()`:
 - Versión de PHP
 - El módulo de json está activado
- Fichero `php.ini`:
 - `extension=json.so`



XAMPP fürfor Windows

Warning: Variable passed to each() is not an array or object in C:\xampp\htdocs\xampp\head.php on line 27

imap

IMAP c-Client Version	2004
SSL Support	enabled

json

json support	enabled
json version	1.2.1

libxml

libXML support	active
libXML Version	? R. 3?
libXML streams	enabled

mbstring

Multibyte Support	enabled
Multibyte string engine	libmbfl
Multibyte (japanese) regex support	enabled
Multibyte regex (oniguruma) version	4.4.4
Multibyte regex (oniguruma) backtrack check	On

Terminado

JSON

- `json_encode()`: codifica un array o un objeto en una cadena JSON
- `json_decode()`: decodifica una cadena en formato JSON y devuelve un array o un objeto
- `json_last_error()`: devuelve el último error ocurrido

JSON

- Ejemplo:

```
$a = array(  
    "España" => array(45, "Madrid", "Barcelona",  
    "Valencia", "Sevilla"),  
    "Estados Unidos" => array(270, "Washington  
DC", "Los Ángeles", "Nueva York", "Chicago"),  
    "México" => array(110, "México DF",  
    "Guadalajara", "Monterrey", "Puebla")  
);  
  
echo json_encode($a);
```




JSON

```
{  
  "España" :  
    [45, "Madrid", "Barcelona", "Valencia", "Sevilla"  
    ],  
  "Estados Unidos": [270, "Washington DC", "Los  
    Ángeles", "Nueva York", "Chicago"],  
  "México" : [110, "México  
    DF", "Guadalajara", "Monterrey", "Puebla"]  
}
```

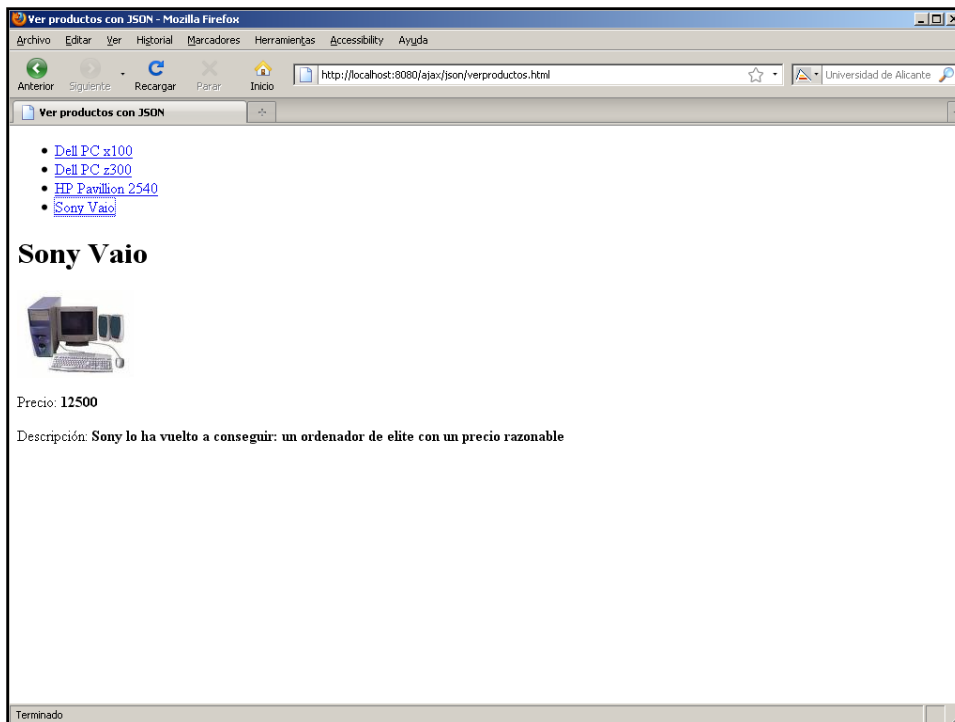
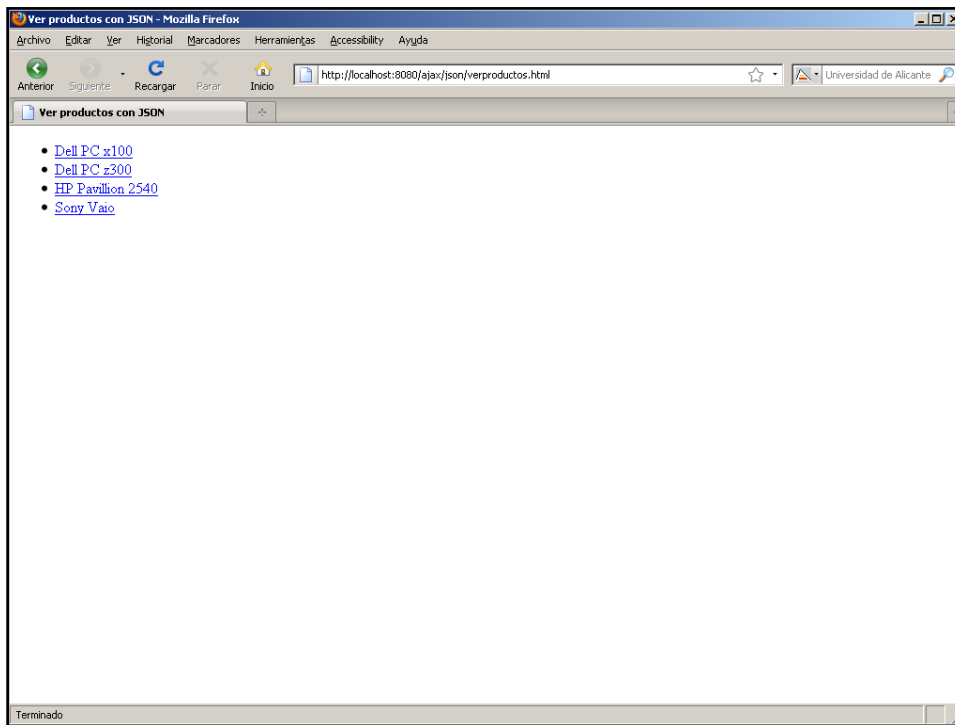


JSON



• EJERCICIO

- Construye un visor de productos mediante AJAX
- Inicialmente se tiene que mostrar sólo la lista de productos (fabricante modelo)
- Cuando el usuario seleccione un producto, se tienen que mostrar todos sus datos: fabricante, modelo, imagen, precio y descripción





JSON

```
<?php
$productos = array(
    // código producto => fabricante, modelo, precio,
    // descripción, imagen
    111 => array("Dell", "PC x100", 8000, "Un gran
ordenador a un gran precio", "ordenador-111.jpg"),
    222 => array("Dell", "PC z300", 15000, "El ultimo
modelo de la familia zXXX", "ordenador-222.jpg"),
    333 => array("HP", "Pavillion 2540", 9300, "Para
los que necesitan la maxima potencia", "ordenador-
333.jpg"),
    444 => array("Sony", "Vaio", 12500, "Sony lo ha
vuelto a conseguir: un ordenador de elite con un precio
razonable", "ordenador-444.jpg"),
);
```



JSON

```
if(!isset($_GET["producto"])) {
    // Devuelve la lista de productos (fabricante
    // modelo)
    foreach($productos as $key => $value)
        $aux[$key] = $value[0] . " " . $value[1];
    $json = json_encode($aux);
}
else
    // Devuelve un producto concreto
    $json =
    json_encode($productos[$_GET["producto"]]);

echo $json;
?>
```



JSON

```
<body onload="listaProductos()">  
  
<div id="productos">  
  
</div>  
  
<div id="producto">  
  
</div>
```



```
function listaProductos()  
{  
  var xmlhttp = ajaxFunction();  
  
  xmlhttp.onreadystatechange = function() {  
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {  
      var htmlProductos = "";  
      var productos = eval("(" + xmlhttp.responseText + ")");  
      for(prod in productos)  
        htmlProductos += '<li><a href="javascript:verProducto(' +  
+ prod + ')";">' + productos[prod] + '</a></li>';  
      document.getElementById("productos").innerHTML = "<ul>" +  
htmlProductos + "</ul>";  
    }  
  }  
  xmlhttp.open("GET", "json-productos.php",true);  
  xmlhttp.send(null);  
}
```



```
function verProducto(prod)
{
  var xmlhttp = ajaxFunction();
  xmlhttp.onreadystatechange = function() {
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
    {
      var htmlProducto = "";
      var producto = eval("(" + xmlhttp.responseText +
      ")");
      htmlProducto += "<h1>" + producto[0] + " " +
      producto[1] + "</h1>";
      htmlProducto += "<p><img src='" + producto[4] +
      "' /></p>";
      htmlProducto += "<p>Precio: <b>" + producto[2] +
      "</b></p>";
      htmlProducto += "<p>Descripción: <b>" +
      producto[3] + "</b></p>";
      document.getElementById("producto").innerHTML =
      htmlProducto;
    }
  }
}
```



XML

- En PHP existen diversas técnicas para leer y escribir un documento XML:
 - A mano
 - Con expresiones regulares
 - DOM (*Document Object Model*)
 - SAX (*Simple API for XML*)

XML

- Independientemente de la técnica empleada, al devolver un documento XML se tiene que indicar el tipo MIME adecuado:

```
<?php
    header("Content-Type: text/xml");
    // O también
    header("Content-Type: application/xml");
?>
```

XML

- DOM:
 - Lee todo el documento XML en memoria
 - Se representa como un árbol de nodos



XML

- DOM lectura:

```
$doc = new DOMDocument();  
  
// Carga un documento desde un fichero  
$doc->load( 'books.xml' );  
// Carga un documento desde una cadena  
$doc->loadXML('<books></books>' );  
  
$books = $doc->getElementsByTagName( "book" );  
$author = $authors->item(0)->nodeValue;
```



XML

- DOM escritura:

```
$doc = new DOMDocument();  
// Salida bonita con sangría y espacios en blanco  
$doc->formatOutput = true;  
$r = $doc->createElement("books");  
$doc->appendChild($r);  
$author = $doc->createElement("author");  
$author->appendChild($doc->createTextNode(  
    $book['author']));  
$b->appendChild($author);  
$doc->save('books.xml');  
echo $doc->saveXML();
```

XML

- SAX:
 - Cada vez que una etiqueta se abre o se cierra o cada vez que se encuentra un texto, se invocan unas funciones
 - Función para etiqueta de apertura
 - Función para etiqueta de cierre
 - Función para manejar los datos
 - No carga el documento XML en memoria → Se puede utilizar con documentos extremadamente grandes

XML

```
$parser = xml_parser_create();  
xml_set_element_handler( $parser,  
    "startElement", "endElement" );  
xml_set_character_data_handler( $parser,  
    "textData" );  
  
$f = fopen( 'books.xml', 'r' );  
  
while($data = fread($f, 4096))  
    xml_parse($parser, $data);  
xml_parser_free($parser);
```


XML

```
function startElement( $parser, $name, $attrs ) {
    global $g_books, $g_elem;
    if ( $name == 'BOOK' )
        $g_books []= array();
    $g_elem = $name; }

function endElement( $parser, $name ) {
    global $g_elem; $g_elem = null; }

function textData( $parser, $text ) {
    global $g_books, $g_elem;
    if ( $g_elem == 'AUTHOR' || $g_elem == 'PUBLISHER' ||
        $g_elem == 'TITLE' )
        $g_books[ count( $g_books ) - 1 ][ $g_elem ] = $text;
}
```

XML



• EJERCICIO

- Transforma el visor de productos mediante AJAX para que utilice XML en vez de JSON
- Ejemplos de documentos XML:



XML

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<productos>
  <producto>
    <codigo>111</codigo>
    <fabricante>Dell</fabricante>
    <modelo>PC x100</modelo>
  </producto>
  <producto>
    <codigo>222</codigo>
    <fabricante>Dell</fabricante>
    <modelo>PC z300</modelo>
  </producto>
  ...
</productos>
```



XML

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<producto>
  <codigo>111</codigo>
  <fabricante>Dell</fabricante>
  <modelo>PC x100</modelo>
  <precio>8000</precio>
  <descripcion>Un gran ordenador a un gran
  precio</descripcion>
  <imagen>ordenador-111.jpg</imagen>
</producto>
```

¿JSON o XML?

- Se suele argumentar que JSON es más abreviado → La cantidad de datos transmitidos es menor
- Sin embargo, esto no se cumple en todas las situaciones

¿JSON o XML?

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



¿JSON o XML?

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



¿JSON o XML?

```
{ "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumbers": [
    { "type": "home", "number": "212 555-1234" },
    { "type": "fax", "number": "646 555-4567" }
  ],
  "newSubscription": false,
  "companyName": null
}
```



¿JSON o XML?

```
<Person firstName="John" lastName="Smith">
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumber type="home">212 555-1234</phoneNumber>
  <phoneNumber type="fax">646 555-4567</phoneNumber>
  <newSubscription>>false</newSubscription>
  <companyName />
</Person>
```



¿JSON o XML?

- Cuando se eliminan los espacios en blanco y saltos de línea, ambos ejemplos ocupan lo mismo
- El beneficio de JSON no es que sea más pequeño a la hora de transmitir, sino que representa mejor la estructura de los datos y requiere menos codificación y procesamiento



¿JSON o XML?

- XML dispone de más herramientas, en la actualidad está integrado en la mayoría de SGBD
- JSON puede ser confuso con datos fuertemente jerarquizados o anidados muy profundamente

