JSON

1) DEFINICIÓN

Con la creciente popularidad de los servicios Web, XML se ha convertido prácticamente de facto en el estándar para transmisión de datos. Pero se necesita transmitir a través de Internet muchos más bytes de información para realizar una tarea que se podría llevar a cabo con un flujo de información mucho más pequeño.

Así se han desarrollado nuevas formas de compresión XML e, incluso, nuevos formatos XML completos, tales como Binary XML (XML binario). Todas estas soluciones funcionan ampliando o añadiéndose a XML, conviniendo los aspectos de compatibilidad descendente en un asunto a tener en cuenta. Douglas Crockford, un experimentado ingeniero software, propuso un nuevo formato de datos construido sobre JavaScript llamado **JSON, JavaScript Object Notation** (notación de objetos JavaScript).

JSON es un formato de datos muy ligero basado en un subconjunto de la sintaxis de JavaScript: literales de matrices y objetos. Como usa la sintaxis JavaScript, las definiciones JSON pueden incluirse dentro de archivos JavaScript y acceder a ellas sin ningún análisis adicional como los necesarios con lenguajes basados en XML.

LITERALES DE MATRIZ

Estos elementos se especifican utilizando corchetes ([y]) para encerrar listas de valores delimitados por comas de JavaScript (lo que puede significar cadenas, números, valores booleanos o valores null) tales como:

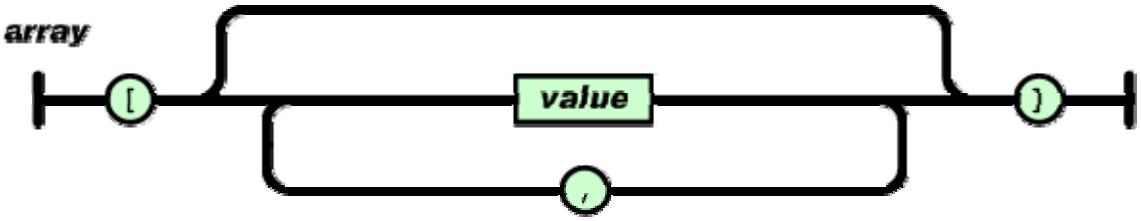
```
var aNombre = ["Jaime", "Pepe", "Alfonso"];
alert (aNombre [0] ) ;          //muestra  "Jaime"
```

OJO: la primera posición de la matriz es 0. Como las matrices en JavaScript no tienen un tipo asignado, pueden utilizarse para almacenar cualquier número o

cualquier tipo de datos diferente.

Si estuviéramos definiendo una matriz sin utilizar la notación literal, tendríamos que utilizar el constructor array. Este sistema no está permitido en JSON.

```
var aValues = new Array("cadena", 24, true, null);
```



LITERALES DE OBJETO

Los literales de objeto se utilizan para almacenar información en parejas nombre-valor para crear un objeto, Un literal de objeto se define mediante llaves ({ y }) Dentro de estas, podemos colocar cualquier número de parejas nombre-valor, definida mediante una cadena, un símbolo de dos puntos y el valor. Cada pareja nombre-valor deben estar separadas por coma.

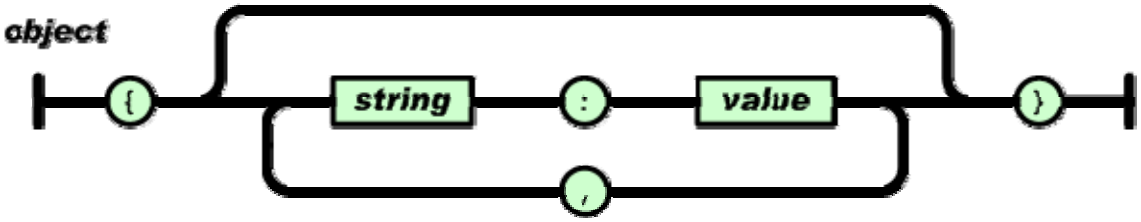
```
var oPersona = {"nombre":"Robert", "edad":30, "hijos":true };
alert(oPersona.nombre);
```

También podemos utilizar la notación de corchetes y pasar el nombre de la propiedad:

```
alert(oPersona["edad"]);
```

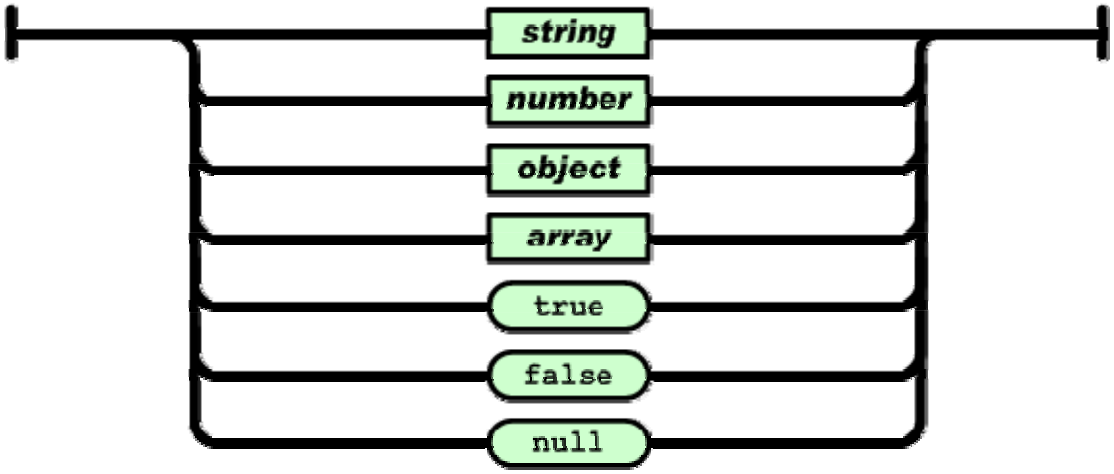
Utilizando el constructor JavaScript Object, que no está permitido en JSON:

```
var oPersona = new Object ();
oPersona.nombre = "red";
...
```



Un valor puede ser una cadena de caracteres con comillas dobles, o un número, o true o false o null, o un objeto o una matriz. Estas estructuras pueden anidarse.

value



Ejemplo:

```
var oPersona2 =
{"nombre":"Robert", "edad":30, "hijos":["Jaime","Pepe","Alfonso"] };
alert(oPersona2.hijos[1]);   //Da Pepe
```

O más complejo:

```
var oPersona3 =
[{"nombre":"Robert", "edad":30, "hijos":["Jaime","Pepe","Alfonso"] },
 {"nombre":"Maria", "edad":36, "hijos":["Hijo Maria","Hijo2 Maria"] }];
alert(oPersona3[1].edad); //Da 36
```

SINTAXIS DE JSON

La sintaxis de JSON realmente no es nada más que la mezcla de literales de objeto y matrices para almacenar datos. JSON representa solamente datos → No incluye el concepto de variables, asignaciones o igualdades.

Este código:

```
var oPersona3 =
[{"nombre":"Robert", "edad":30, "hijos":["Jaime","Pepe","Alfonso"] },
{"nombre":"Maria", "edad":36, "hijos":["Hijo Maria","Hijo2 Maria"] }];
```

Quedaría

```
 [{"nombre":"Robert", "edad":30, "hijos":["Jaime","Pepe","Alfonso"] },
{"nombre":"Maria", "edad":36, "hijos":["Hijo Maria","Hijo2 Maria"] }]
```

Hemos eliminado la variable oPersona3, así como el punto y coma del final. Si transmitimos esta información a través de HTTP a un navegador, será bastante rápido, dado el reducido número de caracteres.

Suponga que recupera esta información utilizando y la almacena en una variable llamada sJSON. Ahora, dispondrá de una cadena de información, no de un objeto. Para transformarla en un objeto, simplemente utilizaremos la función eval ():

```
var sJSON =
'[{"nombre":"Robert", "edad":30, "hijos":["Jaime","Pepe","Alfonso"] },
{"nombre":"Maria", "edad":36, "hijos":["Hijo Maria","Hijo2 Maria"] }]';

var oPersona4   = eval("(" + sJSON + ")");

alert(oPersona4[1].edad);
```

Es importante incluir los paréntesis adicionales alrededor de cualquier cadena JSON antes de pasarla a eval (). Recuerda que las llaves representan también sentencias en JavaScript (como en la sentencia if). La única forma que tiene el intérprete de saber que las llaves representan un objeto y no una sentencia es buscar un signo igual al principio o buscar paréntesis rodeando el texto (lo cual indica que el código es una expresión que se tiene que evaluar en lugar de una sentencia para ejecutar)

2) CODIFICAR Y DECODIFICAR JSON

Como parte de los recursos de JSON, Crockford creó una utilidad que se puede emplear para codificar y descodíficar objetos JSON y JavaScript. El código fuente de esta herramienta se puede descargar en la dirección http://www.json.org/js.html. Por supuesto nosotros no lo vamos a usar, utilizaremos el codificador / decodificador de MOOTOOLS.

Pero, ¿no podíamos decodíficar JSON utilizando la función eval () ?. Hay un problema inherente en la utilización de eval(), que evalúa cualquier código JavaScript que se pasa a la función y esto es un gran riesgo de seguridad.

MOOTOOLS: Json

Json nos facilita la tarea de pasar de un string Json a un Objeto Javascript y viceversa.

Nota: Todo lo referente a JSON respecto a la versión anterior (1.11) ha cambiado bastante, haciendose completamente incompatibles. Por ejemplo, antes se escribía Json. Ahora es JSON.

JSON.encode(objeto)

Nos permite pasar un Objeto Javascript a un simple String.

```
var objeto = {
nombre: 'Mootools',
descripcion: 'Framework'
}

var str = JSON.encode(objeto);
//{"nombre":"Mootools","descripcion":"Framework"}
```

JSON.decode(string, secure)

Nos permite pasar de un String Json a un Objeto Javascript.

```
var str = '{"nombre":"Mootools","descripcion":"Framework"}';
var objeto = JSON.decode(str);
alert(objeto.nombre);            //Mootools
alert(objeto.descripcion);       //Framework
```

Secure es un parámetro opcional que por default es false, pero si lo cambiamos a true, se chequeará la sintaxis del string Json. En caso de que el string sea incorrecto, Json.decode devolverá null.

```
var str = '{"nombre":"Mootools","descripcion":"Framework}';
//Faltan " para Framework
var objeto = JSON.decode(str, true);
if(objeto != null)
alert(objeto.nombre);
else
alert('El string JSON es incorrecto.');
//El string JSON es incorrecto.
```

Request.JSON

Nos permite enviar y recibir objetos JSON mediante AJAX de manera muy simple.

Hereda todos los métodos y propiedades de la clase Request (para trabajo con AJAX) (http://mootools.net/docs/Request/Request).

Recibiendo un objeto:

```
var str_json = new Request.JSON({
    url: "str_json.asp",
    onComplete: function(obj_json) {escribe(obj_json[0].nombre)}
}).get();
```

El resultado será “Susana” si el código de “str_json.asp” es:

```
[{"nombre": "Susana","Edad":36,"Peso": null },
 {"nombre": "Andrea","Edad":25,"Peso": 72 }]
```

Indicamos la url de nuestro script, en este caso str_json.asp. Recibimos el objeto que nos devuelve str_json.asp que es el que viene como parámetro en la función del evento onComplete (obj_json).

También existe onSuccess, que es muy parecido a onComplete. Pero este método también permite recoger la cadena de json.

```
var str_json = new Request.JSON({
  url: "str_json.asp",
  onSuccess: function(obj_json, str_json) {escribe(str_json);}
}).get();
```

El resultado será:

```
[{"nombre": "Susana","Edad":36,"Peso": null },
 {"nombre": "Andrea","Edad":25,"Peso": 72 }]
```

Supongamos que tenemos una base de datos de productos y queremos, mediante ASP y AJAX, mostrar todos esos productos. Con Request.JSON enviaríamos la petición a ASP, éste buscaría todos los productos y los devolvería en formato JSON. Una vez que tenemos los datos en dicho formato, solo debemos mostrarlos con javascript.

Ahora, si no queremos mostrar todos los productos o si deseamos realizar una búsqueda debemos indicarle a ASP cual es el filtro que debe aplicar, y el método get nos permite enviárselo. Este método puede llevar entre paréntesis los valores que le pasamos por get. Por ejemplo:

```
str_json.get({“nombre”:”Susana”})
```

Esto cambiará la url a: str_json.asp?nombre=Susana Ahora con el método get estamos enviando un objeto, el cual puede ser un filtro o un parámetro de búsqueda.

Existe otro método equivalente post. str_json.post({“nombre”:”Susana”})

ACENTOS:
Siempre que trabajemos con AJAX nos vamos a encontrar tarde o temprano con el problema de la codificación de la página. Nosotros trabajos en “ISO-8895-1”, mientras que AJAX toma por defecto “UTF-8” (adiós acentos).

Podríamos pensar que poniendo estas cabeceras en el ASP / PHP al que llamamos mediante AJAX tenemos el problema solucionado. Pero no es así.

ASP

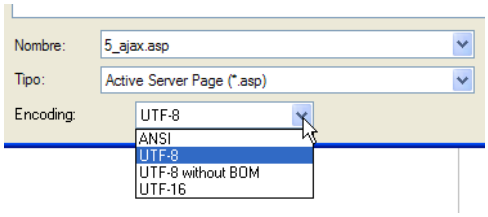
```
<%Response.AddHeader  "charset", "ISO-8859-1"%>
```

PHP

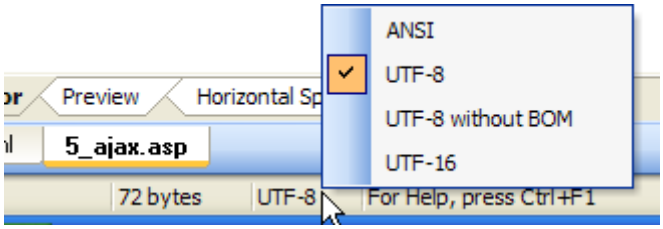
```
<?header(“charset: ISO-8859-1”)?>
```

En teoria Request lleva una propiedad llamada encoding. Pero tampoco funciona.

Solución: Se debe cambiar la codificación del asp de ANSI a UTF-8 al guardar. Desde WebBuilder.



O desde la barra de estado.



Json escrito en ASP

Vamos a conseguir que un ASP "empaquete" correctamente el JSON.

Para ello necesitamos acudir a la página oficial de json.org apartado de ASP http://www.webdevbros.net/2007/04/26/generate-json-from-asp-datatypes/ y descargarnos http://www.webdevbros.net/wp-content/uploads/2008/07/json151.zip

Con la clase que nos hemos descargado de JSON, nos permite pasar desde un ASP:

- Un array
- Un diccionario
- Un registro de una base de datos
- Un objeto

Un array:

JAVASCRIPT

```
    // ------------------------- 3 request sobre un array pasado a json
    var str_json = new Request.JSON({
        url: "str_json1.asp",
        onComplete: function(obj_json) {escribe(obj_json.str_json[0])}
    }).get();
```

Pinta a Felipe.

El código de ASP

```
<!--#include file="json151/json.asp"-->
<%
       dim vector
       vector = array("Felipe",23,true,"")

       set miJson = new JSON
       valores = miJson.toJSON("str_json",vector,false)
       response.write valores
       set miJson = nothing
%>
```

Lo qué pinta este asp:

```
       {"str_json": ["Felipe",23,true,""]}
```

Si pusieramos a true

```
       "str_json": ["Felipe",23,true,""]
```

Un diccionario:

JAVASCRIPT

```
    // ------------------------- 4 request sobre un diccionario pasado a json
    var str_json = new Request.JSON({
        url: "str_json2.asp",
        onComplete: function(obj_json) {escribe(obj_json.str_json.Apellidos)}
    }).get();
```

Pinta a Tango.

El código de ASP

```
<!--#include file="json151/json.asp"-->
<%
	set dic = server.createObject("scripting.dictionary")
	dic.add "Nombre", "Jonatan"
	dic.add "Apellidos", "Tango"
	dic.add "Notas", array(2, 7, 234)

	set miJson = new JSON
	valores = miJson.toJSON("str_json",dic,false)
	response.write valores
	set miJson = nothing
	set dic = nothing
%>
```

Lo qué pinta este asp:

```
{"str_json": {"Nombre": "Jonatan","Apellidos": "Tango","Notas": [2,7,234]}}
```

Un registro de una base de datos

JAVASCRIPT

```
// ------------------------- 5 request sobre una base de datos access
var str_json = new Request.JSON({
  url: "str_json3.asp",
  onComplete: function(obj_json) {escribe(obj_json.str_json[0].apellidos)}
}).get();
```

Pinta Tilla

El código de ASP

```
<!--#include file="json151/json.asp"-->
<%
Set Con = Server.CreateObject( "ADODB.Connection" )
Con.Open "PROVIDER=MICROSOFT.JET.OLEDB.4.0;DATA SOURCE=" &
Server.Mappath("bd1.mdb") & ";"
sql = "select * from personas"
set rs = Con.execute(sql)
set miJson = new JSON
response.write miJson.toJSON("str_json", rs, false)
set miJson = nothing
set Con = nothing
%>
```

Lo qué pinta este asp:

```
{"str_json":
[
{"id": 1,"nombre": "Aitor","apellidos": "Tilla","nota": 1},
{"id": 2,"nombre": "Juanito","apellidos": "Mate","nota": 10},
{"id": 3,"nombre": "Mariano","apellidos": "Ginés","nota": 0}
]}
```

Un objeto

JAVASCRIPT

```
   // ------------------------ 6 request sobre un objeto
    var str_json = new Request.JSON({
        url: "str_json4.asp",
        onComplete: function(obj_json) {escribe(obj_json.str_json.Apellidos)}
    }).get();
```

Pinta Tango

El código de ASP

```
<!--#include file="json151/json.asp"-->
<%
class persona
    public nombre, apellidos, notas

    public function reflect()
        set reflect = server.createObject("scripting.dictionary")
        reflect.add "Nombre", nombre
        reflect.add "Apellidos", apellidos
        reflect.add "Notas", notas
    end function
end class

set p = new persona
p.nombre = "Jonatan"
p.apellidos = "Tango"
p.notas = array(2, 7, 234)

set miJson = new JSON
valores = miJson.toJSON("str_json",p,false)
response.write valores
set miJson = nothing
set p = nothing
%>
```

Podemos observar el método “reflect” que es especial para que la clase de ASP JSON funcione.

Lo qué pinta este asp:

```
{"str_json": {"Nombre": "Jonatan","Apellidos": "Tango","Notas": [2,7,234]}}
```

Json leido desde ASP

Descargamos el componente que nos hace falta desde:
http://www.crayoncowboy.com/downloads/asp/CCBJSONParser.zip

Información detallada de este componente la tenemos en
http://blog.crayoncowboy.com/?p=7

Supongamos que nos llega esta cadena

```
str_json = "{'Susana':{'Correo' : 'pp@uno.es'}, 'Juan' : {'Correo' :
'pp@uno.es'}}"
```

Que la recibe por post, get o simplemente la escribimos.

```
<!--#include file="json151/CCB.JSONParser.asp"-->
<%
set JPar = new CCBJsonParser
str_json = "{'Susana':{'Correo' : 'pp@uno.es'}, 'Juan' : {'Correo' :
'pp@uno.es'}}"
JPar.JSON = str_json
correo = JPar.valueFor("Susana.Correo")
response.write "Susana.Nombre  "  & correo & "<hr />"

JPar.parse
set personal = JPar.dictionary
for each personas in personal.keys
    Response.Write personas & " " & personal.item(personas).item("Correo") &
"<br />"
neXt

set personal = nothing
set Jpar = nothing

response.end
%>
```

Hacemos referencia a la librería descargada:

```
<!--#include file="json151/CCB.JSONParser.asp"-->
```

Creamos objeto:

```
set JPar = new CCBJsonParser
```

Y luego o localizamos el valor de un elemento.propiedad (Susana.Correo) cuyo resultado será “Susana.Nombre pp@uno.es”.

O podemos crear un diccionario :

```
JPar.parse
set personal = JPar.dictionary
```

Que se comporta como un diccionario ASP.

```
Response.write personal.item("Susana").item("Correo")
```

Ejercicios

4.- Partiendo de la base de datos que se te entrega “asignturas.mdb” y con esta estructura de datos

	Nombre del campo	Tipo de datos
🔑	id	Número
	nombre	Texto
	basico	Sí/No
	caca	Texto

y con estos datos:

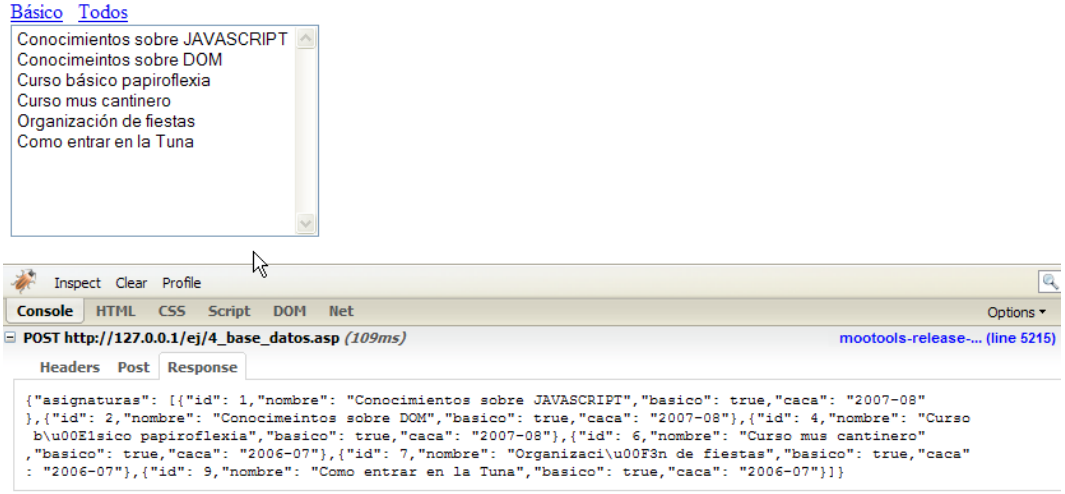
	id	nombre	basico	caca
▶	1	Conocimientos sobre JAVASCRIPT	☑	2007-08
	2	Conocimeintos sobre DOM	☑	2007-08
	3	Conocimientos sobre MOOTOOLS	☐	2007-08
	4	Curso básico papiroflexia	☑	2007-08
	5	Curso avanzado aviones papel	☐	2007-08
	6	Curso mus cantinero	☑	2006-07
	7	Organización de fiestas	☑	2006-07
	8	Como entrar en la Tuna y no repetir cur	☐	2006-07
	9	Como entrar en la Tuna	☑	2006-07

se pide crear una página html que tenga dos enlaces: básico y todos

Al pinchar sobre “básico” debe aparecer una lista de opciones de los cursos básicos.
Al pinchar sobre “todos” debe aparecer una lista de opciones de todos los cursos.

Nunca se debe recargar la página. Además la información debe de salir de esa base de datos, de manera que si se actualiza los listados se deben actualizar también.

Básico:



Todos:

Básico Todos

- Conocimientos sobre JAVASCRIPT
- Conocimeintos sobre DOM
- Conocimientos sobre MOOTOOLS
- Curso básico papiroflexia
- Curso avanzado aviones papel
- Curso mus cantinero
- Organización de fiestas
- Como entrar en la Tuna y no repetir curso
- Como entrar en la Tuna

Inspect Clear Profile

Console HTML CSS Script DOM Net Options

POST http://127.0.0.1/ej/4_base_datos.asp *(109ms)* mootools-release-... (line 5215)

POST http://127.0.0.1/ej/4_base_datos.asp *(141ms)* mootools-release-... (line 5215)

Headers Post Response


```
{"asignaturas": [{"id": 1,"nombre": "Conocimientos sobre JAVASCRIPT","basico": true,"caca": "2007-08"
},{"id": 2,"nombre": "Conocimeintos sobre DOM","basico": true,"caca": "2007-08"},{"id": 3,"nombre": "Conocimientos
 sobre MOOTOOLS","basico": false,"caca": "2007-08"},{"id": 4,"nombre": "Curso b\u00E1sico papiroflexia"
,"basico": true,"caca": "2007-08"},{"id": 5,"nombre": "Curso avanzado aviones papel","basico": false
,"caca": "2007-08"},{"id": 6,"nombre": "Curso mus cantinero","basico": true,"caca": "2006-07"},{"id"
: 7,"nombre": "Organizaci\u00F3n de fiestas","basico": true,"caca": "2006-07"},{"id": 8,"nombre": "Como
 entrar en la Tuna y no repetir curso","basico": false,"caca": "2006-07"},{"id": 9,"nombre": "Como entrar
 en la Tuna","basico": true,"caca": "2006-07"}]}
```