INSTITUTO SUPERIOR TECNOLOGICO DE TURISMO Y PATRIMONIO "YAVIRAC"

Nombre: jean Pierre Arias **Curso:** 1"B" **Fecha:** 17/09/2018

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 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 ();
o Persona.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 unidarse.

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 Æ No incluye el concepto de variables, asignaciones o igualdades.

Este es 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 o Persona 3, 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 }]
```