**UNIVERSIDAD TECNOLÓGICA DE PEREIRA**

**FACULTAD DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN**

**INVESTIGACION: JSON**

**ESTRUCTURA DE DATOS**

**PRESENTADO POR:**

José David Melo

**DOCENTE:**

Ricardo Antonio Bermúdez Osorio

**UNIVERSIDAD TECNOLÓGICA DE PEREIRA**

**Jornada Especial**

**2017 – II**

**JSON: JavaScript Object Notation**

**JSON**, acrónimo de *JavaScript Object Notation*, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript, aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que es mucho más sencillo escribir un analizador sintáctico(parser) de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval(), lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

**Constituido por dos estructuras:**

* Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
* Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

**Sintaxis:**

Los tipos de datos disponibles con JSON son:

* Números: Se permiten números negativos y opcionalmente pueden contener parte fraccional separada por puntos. Ejemplo: 123.456
* Cadenas: Representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape. Ejemplo: "Hola"
* Booleanos: Representan valores booleanos y pueden tener dos valores: true y false
* null: Representan el valor nulo.
* Array: Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes. Ejemplo ["juan","pedro","jacinto"]
* Objetos: Son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves. El nombre tiene que ser una cadena y entre ellas. El valor puede ser de cualquier tipo. Ejemplo: {"departamento":8,"nombredepto":"Ventas","director": "juan rodriguez","empleados":[{"nombre":"Pedro","apellido":"Fernandez"},{"nombre":"Jacinto","apellido":"Benavente"} ]}

**Modelos de procesamiento**

Al ser JSON un formato muy extendido para el intercambio de datos, se han desarrollado API para distintos lenguajes (por ejemplo, ActionScript, C, C++, C#, ColdFusion, Common Lisp, Delphi, E, Eiffel, Java, JavaScript, ML, Objective-C, Objective CAML, Perl, PHP, Python, Rebol, Ruby, Lua y Visual FoxPro) que permiten analizar sintácticamente, generar, transformar y procesar este tipo de datos.

Los modelos de programación más utilizados para tratar con JSON en los distintos lenguajes son:2​

**Modelo de objeto**. - El JSON completo es almacenado en memoria en un formato de árbol. Este árbol es navegado, analizado y modificado con las API apropiadas. Como lo carga todo en memoria y luego lo procesa este modelo consume muchos recursos. Sin embargo, es muy flexible para manipular el contenido. Este modelo es permitido por ejemplo en java por la JSR 353 y por la biblioteca Jackson

**Modelo de flujo**: Los datos son leídos o escritos en bloques. Por ejemplo, cada vez que se lee un bloque el analizador genera eventos apropiados para indicar el tipo de bloque de que se trata. El cliente puede procesar el contenido escuchando los eventos apropiados. Además, es el cliente el que decide cómo se va leyendo el JSON permitiendo parar o saltar contenidos en mitad del proceso. El proceso de escritura tiene propiedades análogas. Por ejemplo, este modelo es permitido en java por la JSR 353.

Convirtiendo los objetos JSON en objetos del lenguaje. En java esto es realizado por ejemplo por las bibliotecas Jackson y Gson.

**Uso de JSON**

En teoría, es trivial analizar JSON en JavaScript usando la función JSON.parse() incorporada en el lenguaje. Por ejemplo:

miObjeto = JSON.parse(json\_datos);

En la práctica, las consideraciones de seguridad por lo general recomiendan no usar eval sobre datos crudos y debería usarse un analizador JavaScript distinto para garantizar la seguridad. El analizador proporcionado por JSON.org usa eval() en su función de análisis, protegiéndola con una expresión regular de forma que la función sólo ve expresiones seguras.

Un ejemplo de acceso a datos JSON usando XMLHttpRequest es:

**var** http\_request = **new** XMLHttpRequest();

**var** url = "http://example.net/jsondata.php"; *// Esta URL debería devolver datos JSON*

*// Descarga los datos JSON del servidor.*

http\_request.onreadystatechange = handle\_json;

http\_request.open("GET", url, **true**);

http\_request.send(**null**);

**function** handle\_json() {

**if** (http\_request.readyState == 4) {

**if** (http\_request.status == 200) {

**var** json\_data = http\_request.responseText;

**var** the\_object = eval("(" + json\_data + ")");

} **else** {

alert("Ocurrio un problema con la URL.");

}

http\_request = **null**;

}

}

Obsérvese que el uso de XMLHttpRequest en este ejemplo no es compatible con todos los navegadores, ya que existen variaciones sintácticas para Internet Explorer, Opera, Safari, y navegadores basados en Mozilla.

También es posible usar elementos <iframe> ocultos para solicitar los datos de manera asíncrona, o usar peticiones <form target="url\_to\_cgi\_script" />. Estos métodos eran los más habituales antes del advenimiento del uso generalizado de XMLHttpRequest.

Hay una biblioteca[3](https://es.wikipedia.org/wiki/JSON#cite_note-3)​ para el framework .NET que exporta clases .NET con la sintaxis de JSON para la comunicación entre cliente y servidor, en ambos sentidos.

**Ejemplo de JSON**

A continuación, se muestra un ejemplo simple de definición de barra de menús usando JSON y XML.

JSON:

{

"menu": {

"id": "file",

"value": "File",

"popup": {

"menuitem": [

{

"value": "New", "onclick": "CreateNewDoc()"

},{

"value": "Open", "onclick": "OpenDoc()"

},{

"value": "Close", "onclick": "CloseDoc()"

}

]

}

}

}

Es una posible representación JSON del siguiente XML:

**<menu** id="file" value="File"**>**

**<popup>**

**<menuitem** value="New" onclick="CreateNewDoc()" **/>**

**<menuitem** value="Open" onclick="OpenDoc()" **/>**

**<menuitem** value="Close" onclick="CloseDoc()" **/>**

**</popup>**

**</menu>**

**Comparación con XML y otros lenguajes de marcado**

XML goza de mucho menor soporte y ofrece menos herramientas de desarrollo (tanto en el lado del cliente como en el lado del servidor). Hay muchos analizadores JSON en el lado del servidor, existiendo al menos un analizador para la mayoría de los entornos. En algunos lenguajes, como Java o PHP, hay diferentes implementaciones donde escoger. En JavaScript, el análisis es posible de manera nativa con la función JSON.parse().  Ambos formatos carecen de un mecanismo para representar grandes objetos binarios.

Con independencia de la comparación con XML, JSON puede ser muy compacto y eficiente si se usa de manera efectiva. Por ejemplo, la aplicación DHTML de búsqueda en «BarracudaDrive» (en inglés). Archivado desde el original el 1 de diciembre de 2015. recibe los listados de directorio como JSON desde el servidor. Esta aplicación de búsqueda está permanentemente consultando al servidor por nuevos directorios, y es notablemente rápida, incluso sobre una conexión lenta.

Los entornos en el servidor normalmente requieren que se incorpore una función u objeto analizador de JSON. Algunos programadores, especialmente los familiarizados con el lenguaje C, encuentran JSON más natural que XML, pero otros desarrolladores encuentran su escueta notación algo confusa, especialmente cuando se trata de datos fuertemente jerarquizados o anidados muy profundamente.

Hay más comparaciones entre JSON y XML en JSON.org4​

YAML es un supe conjunto de JSON que trata de superar algunas de las limitaciones de éste. Aunque es significativamente más complejo,5​ aún puede considerarse como ligero. El lenguaje de programación Ruby utiliza YAML como el formato de serialización por defecto. Así pues, es posible manejar JSON con bastante sencillez.

**REFERENCIA:**

<https://es.wikipedia.org/wiki/JSON>

<http://www.json.org/json-es.html>