

**Módulo:**

Fundamentos de Ingeniería de Software

**Segmento:**

Algoritmos y Estructuras de Datos

**Tema:**

JSON

Alumno: Sebastian Nazarian

[sebanazarian@gmail.com](mailto:sebanazarian@gmail.com)

**Tabla de contenido**

[JSON 3](#_Toc535026211)

[1. Biblioteca json.org en Java 3](#_Toc535026212)

[1.1. Instalación 3](#_Toc535026213)

[1.2. Crear JSO usando org.json 4](#_Toc535026214)

[1.3. Analizar JSON usando org.json 6](#_Toc535026215)

[2. Usando JSON.NET Library 9](#_Toc535026216)

[2.1. Beneficios y características 9](#_Toc535026217)

[2.2. JsonConvert 9](#_Toc535026218)

[2.3. JsonSerializer 10](#_Toc535026219)

[2.4. Ejemplo de Lectura con JsonTextReader 10](#_Toc535026220)

[2.5. Ejemplo de escritura con JsonTextWriter 11](#_Toc535026221)

[3. Bibliografia 12](#_Toc535026222)

## JSON

[JSON](http://json.org/) (JavaScript Object Notation) es un formato para intercambio de datos liviano, basado en texto e independiente del lenguaje de programación, que resulta fácil de escribir y leer tanto para los seres humanos como para las máquinas. JSON puede representar dos tipos estructurados: *objetos* y *matrices*. Un objeto es una colección no ordenada de cero o más pares de nombres/valores. Una matriz es una secuencia ordenada de cero o más valores. Los valores pueden ser cadenas, números, booleanos, nulos y estos dos tipos estructurados.

JSON está 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 sequencias.

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.

## Biblioteca json.org en Java

### Instalación

La biblioteca JSON-Java (JSON en Java) también se conoce como org.json.

La implementación de org.json ya se ha incluido en Android SDK. Así que puedes usarlo sin ninguna instalación adicional en proyectos de Android.

Tiene dos formas opcionales de instalar org.json en un proyecto Java.

* Simplemente descargue el archivo jar de org.json, luego péguelo en la carpeta del proyecto y agréguelo a la ruta de compilación del proyecto.
* Si su proyecto es un proyecto de Maven, simplemente agregue una dependencia al archivo pom.xml de Maven de su proyecto.

<!-- pom.xml -->

<dependencies>

<!-- ... -->

<dependency>

<groupId>org.json</groupId>

<artifactId>json</artifactId>

<version>20160810</version>

</dependency>

<!-- ... -->

</dependencies>

### Crear JSO usando org.json

#### Crea JSON directamente:

org.json utiliza su clase JSONObject (Java Doc) para crear o analizar JSON. Las API de JSONObject funcionan de manera muy similar a las API de Java Map y son fáciles de usar.

import org.json.JSONObject;

private static void createJSON(boolean prettyPrint) {

JSONObject tomJsonObj = new JSONObject();

tomJsonObj.put("name", "Tom");

tomJsonObj.put("birthday", "1940-02-10");

tomJsonObj.put("age", 76);

tomJsonObj.put("married", false);

// Cannot set null directly

tomJsonObj.put("car", JSONObject.NULL);

tomJsonObj.put("favorite\_foods", new String[] { "cookie", "fish", "chips" });

// {"id": 100001, "nationality", "American"}

JSONObject passportJsonObj = new JSONObject();

passportJsonObj.put("id", 100001);

passportJsonObj.put("nationality", "American");

// Value of a key is a JSONObject

tomJsonObj.put("passport", passportJsonObj);

if (prettyPrint) {

// With four indent spaces

System.out.println(tomJsonObj.toString(4));

} else {

System.out.println(tomJsonObj.toString());

}

}

La cadena JSON de salida se lista de la siguiente manera.

{

"car": null,

"birthday": "1940-02-10",

"married": false,

"age": 76,

"name": "Tom",

"favorite\_foods": [

"cookie",

"fish",

"chips"

],

"passport": {

"id": "100001",

"nationality": "American"

}

}

Consejos: ¿cómo establecer el valor de una clave en nulo en org.json.JSONObject?

Debe establecer el valor en JSONObject.NULL en lugar del propio nulo de Java.

Consejos: ¿cómo decirle a JSONObject que genere una cadena JSON de impresión bonita?

Un método de sobrecarga de toString () de JSONObject puede aceptar un argumento entero que significa el número de espacios a sangrar.

tomJsonObj.toString (4);

#### Crear JSON desde el objeto Java Map

JSONObject proporcionó un constructor para convertir datos de pares clave-valor a objeto JSON desde el objeto Java Map.

mport org.json.JSONObject;

private static void createJSONFromMap(boolean prettyPrint) {

// Java Map object to store key-value pairs

Map<String, Object> tom = new HashMap<String, Object>();

tom.put("name", "Tom");

tom.put("birthday", "1940-02-10");

tom.put("age", 76);

tom.put("married", false);

// Must be JSONObject.NULL instead of null

tom.put("car", JSONObject.NULL);

tom.put("favorite\_foods", new String[] { "cookie", "fish", "chips" });

Map<String, Object> passport = new HashMap<String, Object>();

passport.put("id", 100001);

passport.put("nationality", "American");

tom.put("passport", passport);

// Create JSON object from Java Map

JSONObject tomJsonObj = new JSONObject(tom);

if (prettyPrint) {

// With 4 indent spaces

System.out.println(tomJsonObj.toString(4));

} else {

System.out.println(tomJsonObj.toString());

}

}

### Analizar JSON usando org.json

#### Json para analizar

{

"name": "Tom",

"birthday": "1940-02-10",

"age": 76,

"married": false,

"car": null,

"favorite\_foods": [

"cookie",

"chips",

"fish"

],

"passport": {

"id": 100001,

"nationality": "American"

}

}

import java.io.File;

import org.apache.commons.io.FileUtils;

import org.json.JSONObject;

private static void readJSON() throws Exception {

File file = new File("./tom.json");

String content = FileUtils.readFileToString(file, "utf-8");

// Convert JSON string to JSONObject

JSONObject tomJsonObject = new JSONObject(content);

}

#### Recuperar el nombre

String name = tomJsonObject.getString("name");

#### Recuperar la edad

int age = tomJsonObject.getInt("age");

#### Obtener el array de comidas favoritas

JSONArray favorite\_foods = tomJsonObject.getJSONArray("favorite\_foods");

for (int i = 0; i < favorite\_foods.length(); i++) {

String food = (String) favorite\_foods.get(i);

System.out.println(food);

}

// Or convert the JSONArray to Java List

List<Object> foods = favorite\_foods.toList();

for (Object food : foods) {

System.out.println((String)food);

}

#### Obtener un valor booleano:

boolean married = tomJsonObject.getBoolean("married");

#### Acceder a un diccionario

JSONObject passportJsonObject = tomJsonObject.getJSONObject("passport");

String nationality = passportJsonObject.getString("nationality");

#### Manejos de Null

Consejos: ¿cómo manejar la clave inexistente o el valor nulo (JSONObject.NULL) en JSONObject?

Por ejemplo, en el archivo JSON de muestra, la clave car que existe pero el valor es nula (JSONObject.NULL); La clave houseno existe en absoluto.

Si su requisito es solo verificar la existencia de la clave especificada, el método has (String key) será su elección.

tomJsonObject.has("car"); // true, "car" key exists even the value is JSONObject.NULL

tomJsonObject.has("house"); // false, the "house" key doesn't exist

Como se mencionó anteriormente, el nulo en JSON es totalmente diferente con el propio nulo de Java en la vista de JSONObject. JSONObject usa una constante llamada JSONObject.NULL para representar el nulo en JSON. Así que a veces es posible que tenga que verificar estos dos escenarios respectivamente.

/\* Key exists but value is null (JSONObject.NULL) \*/

tomJsonObject.opt("car") == JSONObject.NULL; // true

tomJsonObject.opt("car") == null; // false, JSONObject.NULL is not the same as Java's own null

/\* The key doesn't exist at all \*/

tomJsonObject.opt("house") == JSONObject.NULL; // false

tomJsonObject.opt("house") == null; // true

String car = tomJsonObject.optString("car", "No car!");

System.out.println(car); // "No car!"

String house = tomJsonObject.optString("house", "No house!");

System.out.println(house); // "No house!"

## Usando JSON.NET Library

Json.NET es un popular marco JSON de alto rendimiento para .NET

### Beneficios y características

* Flexible serializador JSON para convertir entre objetos .NET y JSON
* LINQ to JSON para leer y escribir manualmente JSON
* Alto rendimiento: más rápido que los serializadores JSON incorporados de .NET
* Escribe JSON con sangría y fácil de leer
* Convertir JSON desde y hacia XML
* Es compatible con [.NET Standard 2.0](https://github.com/dotnet/standard/blob/master/docs/versions.md) , .NET 2, .NET 3.5, .NET 4, .NET 4.5, Silverlight, Windows Phone y Windows 8 Store

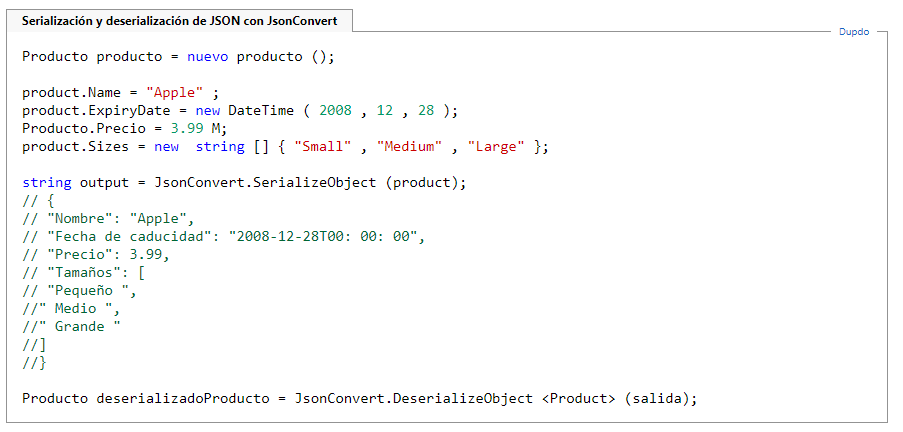
El serializador JSON en Json.NET es una buena opción cuando el JSON que está leyendo o escribiendo se acerca a una clase .NET.

LINQ to JSON es bueno para situaciones en las que solo está interesado en obtener valores de JSON, no tiene una clase para serializar o deserializar, o JSON es radicalmente diferente de su clase y necesita leer y escribir manualmente desde su objetos.

El método más rápido de conversión entre texto JSON y un objeto .NET es usar el [JsonSerializer](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonSerializer.htm) . El JsonSerializer convierte los objetos .NET en su equivalente JSON y vuelve a asignar los nombres de las propiedades del objeto .NET a los nombres de las propiedades JSON y copia los valores por usted.

### JsonConvert

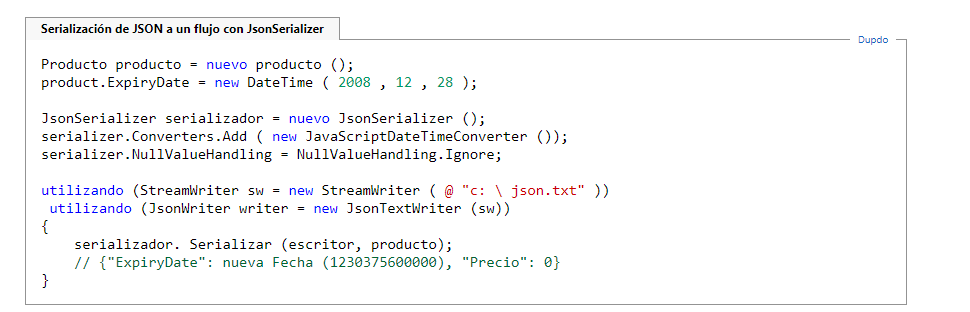
Para los escenarios simples en los que desea convertir desde y hacia una cadena JSON, los métodos [SerializeObject ()](https://www.newtonsoft.com/json/help/html/Overload_Newtonsoft_Json_JsonConvert_SerializeObject.htm) y[DeserializeObject ()](https://www.newtonsoft.com/json/help/html/Overload_Newtonsoft_Json_JsonConvert_DeserializeObject.htm) en JsonConvert proporcionan un envoltorio fácil de usar sobre JsonSerializer.



SerializeObject y DeserializeObject tienen sobrecargas que toman un objeto [JsonSerializerSettings](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonSerializerSettings.htm) . JsonSerializerSettings le permite usar muchas de las configuraciones de JsonSerializer que se enumeran a continuación mientras sigue usando los métodos simples de serialización.

### JsonSerializer

Para tener más control sobre cómo se serializa un objeto, se puede usar [JsonSerializer](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonSerializer.htm) directamente. El JsonSerializer puede leer y escribir texto JSON directamente en un flujo a través de [JsonTextReader](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonTextReader.htm) y [JsonTextWriter](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonTextWriter.htm) . También se pueden usar otros tipos de JsonWriters, como [JTokenReader](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JTokenReader.htm) / [JTokenWriter](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Linq_JTokenWriter.htm) , para convertir su objeto desde y hacia LINQ a objetos JSON, o [BsonReader](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Bson_BsonReader.htm) / [BsonWriter](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_Bson_BsonWriter.htm) , para convertir hacia y desde BSON.



JsonSerializer tiene una serie de propiedades para personalizar cómo serializa JSON. Estos también se pueden usar con los métodos en JsonConvert a través de las sobrecargas de JsonSerializerSettings.

### Ejemplo de Lectura con JsonTextReader

cadena json = @ "{

'CPU': 'Intel',

'PSU': '500W',

'Drives': [

'DVD leído / escritor'

/\*(roto)\*/,

'Disco duro de 500 gigabytes',

'Disco duro de 200 gigabytes'

]

} " ;

JsonTextReader reader = new JsonTextReader ( new StringReader (json));

mientras que (reader.Read ())

{

if (reader.Value! = null )

{

Console.WriteLine ( "Token: {0}, Valor: {1}" , reader.TokenType, reader.Value);

}

más

{

Console.WriteLine ( "Token: {0}" , reader.TokenType);

}

}

// Token: StartObject

// Token: PropertyName, Valor: CPU

// Token: String, Valor: Intel

// Token: PropertyName, Value: PSU

// Token: String, Valor: 500W

// Token: PropertyName, Value: Drives

// Token: StartArray

// Token: String, Valor: DVD leído / grabador

// Token: Comentario, Valor: (roto)

// Token: String, Valor: 500 gigabyte disco duro

// Token: String, Valor: 200 gigabyte disco duro

// Token: EndArray

// Token: EndObject

### Ejemplo de escritura con JsonTextWriter

StringBuilder sb = new StringBuilder ();

StringWriter sw = new StringWriter (sb);

utilizando (escritor JsonWriter = nuevo JsonTextWriter (sw))

{

writer.Formatting = Formatting.Indented;

writer.WriteStartObject ();

writer.WritePropertyName ( "CPU" );

writer.WriteValue ( "Intel" );

writer.WritePropertyName ( "PSU" );

writer.WriteValue ( "500W" );

writer.WritePropertyName ( "Drives" );

writer.WriteStartArray ();

writer.WriteValue ( "DVD read / writer" );

writer.WriteComment ( "(roto)" );

writer.WriteValue ( "disco duro de 500 gigabytes" );

writer.WriteValue ( "disco duro de 200 gigabytes" );

writer.WriteEnd ();

writer.WriteEndObject ();

}

Console.WriteLine (sb.ToString ());

// {

// "CPU": "Intel",

// "PSU": "500W",

// "Drives": [

// "DVD read / writer"

// / \* (roto) \* /,

// "Disco duro de 500 gigabytes",

// "Disco duro de 200 gigabytes"

//]

//}

## Bibliografia

<https://www.newtonsoft.com/json/help/html/Introduction.htm>

<https://www.codevoila.com/post/65/java-json-tutorial-and-example-json-java-orgjson>