2

Bibliotecas JSON para JAVA

1. Introducción

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos que se utiliza ampliamente en la comunicación asíncrona entre servidores y navegadores web. JSON expresa datos como pares clave-valor legibles por humanos, lo que lo hace fácil de entender y manipular.

Aunque los navegadores pueden interpretar JSON de manera nativa, en el contexto de Java (tanto en el servidor como en el cliente), es necesario utilizar bibliotecas específicas para **analizar** y **generar** JSON.

Características del JSON:

- Es independiente del lenguaje y se utiliza ampliamente en sistemas de bases de datos NoSQL como MongoDB y CouchDB.
- Es más sencillo de leer y manipular que XML, lo que lo convierte en una opción preferida para muchos desarrolladores.

2. APIs de JSON en Java

Java, inicialmente, no tenía una implementación estándar para trabajar con JSON, por lo que surgieron varias bibliotecas y APIs de código abierto para facilitar este trabajo. A continuación se presentan algunas de las bibliotecas más populares y sus características:

2.1 GSON

GSON es una biblioteca de **Google** para trabajar con JSON en Java. GSON permite convertir objetos Java a JSON y viceversa, y proporciona flexibilidad en su uso. Actualmente, se considera una de las bibliotecas más completas y rápidas.

Características:

- Analiza y genera JSON.
- ▼ Utiliza clases como `Gson`, `JsonReader`, y `JsonParser`.
  - La clase `Gson` que **puede analizar objetos JSON en objetos Java personalizados y viceversa**, a través de los métodos `fromJSon` y `toJson`, respectivamente.

- El `JsonReader`, que es el analizador JSON de flujos de GSON, que **analiza un token JSON a la vez**.
- El `JsonParser` que puede **analizar JSON** en una estructura de árbol de objetos Java específicos de GSON.

Dependencia Maven:

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.11.0</version>
</dependency>
```

2.2 Jackson

Jackson es otra biblioteca muy popular para trabajar con JSON en Java, con un enfoque similar al de GSON. Ofrece varias opciones para analizar y generar JSON, como el `ObjectMapper` y el `JsonParser`.

Características:

- Analiza y genera JSON.
- ▼ Utiliza clases como `ObjectMapper` , `JsonParser`, y `JsonGenerator` .
  - **Jackson** `ObjectMapper` para deserializar y serializar objetos Java.
  - **Jackson** `JsonParser`, que es el **analizador de extracción JSON de Jackson**, analizando JSON un token a la vez.
  - **Jackson** `JsonGenerator` que puede generar JSON un token a la vez.

Dependencia Maven:

```
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
```

```
    <version>2.15.3</version>
</dependency>
```

Ejemplo de uso:

```
ObjectMapper mapper = new ObjectMapper();
Alumno objeto = new Alumno(4, "Otto");

// Conversión en JSON (serialización):
String jsonStr = mapper.writeValueAsString(objeto);

// Lectura de objeto JSON:
Alumno alumno = mapper.readValue(jsonStr, Alumno.class);
```

2.3 JSONP: Jakarta JSON Processing

JSON-P es una especificación que proporciona una API para procesar documentos JSON en Java. Se usa principalmente en aplicaciones Java EE, pero también es aplicable en Java SE.

Características:

- Proporciona un modelo de objeto JSON similar a un árbol y una API de streaming basada en eventos.
- Permite la manipulación y análisis de documentos JSON.

Dependencia Maven:

```
<dependency>
    <groupId>jakarta.json</groupId>
    <artifactId>jakarta.json-api</artifactId>
    <version>2.1.2</version>
</dependency>
```

Ejemplo de uso:

```
import jakarta.json.*;
import java.io.StringWriter;

public class JSonPExemplo {

    public static void main(String[] args) {
        // Crear un objeto JSON usando JSON-P
        JsonObject objetoJson = Json.createObjectBuilder()
                .add("nombre", "Otto")
                .add("edad", 4)
                .add("ciudad", "Santiado de Compostela")
                .build();

        // Convertir el objeto JSON a una cadena
        StringWriter stringWriter = new StringWriter();
        try (JsonWriter jsonWriter = Json.createWriter(stringWriter)) {
            jsonWriter.writeObject(objetoJson);
        }

        // Imprimir la cadena JSON
        String strJson = stringWriter.toString();
        System.out.println("JSON Resultante (JSON-P):");
        System.out.println(strJson);
    }
}
```

2.4 JSON-B: Jakarta JSON Binding

JSON-B es una API que simplifica la conversión de objetos Java a JSON y viceversa. A diferencia de JSON-P, que se centra en el procesamiento de JSON, JSON-B se enfoca en la serialización y deserialización automática.

Características:

- Utiliza anotaciones como `@JsonbProperty` y `@JsonbTransient` para personalizar el mapeo entre Java y JSON.
- Simplifica la conversión mediante un enfoque de más alto nivel.

Dependencia Maven:

```
<dependency>
    <groupId>javax.json.bind</groupId>
    <artifactId>javax.json.bind-api</artifactId>
    <version>3.0.0</version>
</dependency>

<dependency>
    <groupId>org.eclipse</groupId>
    <artifactId>yasson</artifactId>
    <version>3.0.3</version>
</dependency>
```

Ejemplo de uso:

```
import jakarta.json.bind.Jsonb;
import jakarta.json.bind.JsonbBuilder;

public class JsonBExemplo {

    public static void main(String[] args) {
        // Crear un objeto Java
        Persona persona = new Persona("Otto", 4, "Santiago de Compostela");

        // Crear un objeto Jsonb
        Jsonb jsonb = JsonbBuilder.create();

        // Convertir el objeto Java a JSON
        String strJson = jsonb.toJson(persona);

        // Imprimir la cadena JSON
        System.out.println("JSON Resultante (JSON-B):");
        System.out.println(strJson);
    }
}
```

2.5 JSON.org

JSON.org ofrece una biblioteca de código abierto que fue una de las primeras en estar disponible, pero no es tan flexible ni rápida como otras alternativas actuales como GSON o Jackson.

2.6 Otras Bibliotecas

- **mJson** (Descontinuado): Ofrecía soporte completo para validación de JSON Schema Draft 4.
- **Boon** (Descontinuado): Se decía que era una de las más rápidas, pero ha sido descontinuada.

3. Comparación de Bibliotecas

Rendimiento:

Un análisis de rendimiento reciente muestra que GSON es la biblioteca más rápida y flexible en comparación con otras como Jackson y JSON-P.

Ejemplo de resultados de rendimiento (parsing):

Biblioteca	Velocidad (MB/ms)	Tiempo de Parsing (%)
GSON	100%	0%
Jackson	58%	70.87%
JSON.simple	79%	126.58%
JSONP	44%	25.49%

¿Cuál es mejor?

- **JSON-B** es preferido cuando se requiere una fácil conversión entre objetos Java y JSON, especialmente en aplicaciones que usan Java EE.
- **GSON** es la opción más flexible y rápida para la mayoría de los proyectos pequeños y medianos.
- **Jackson** sigue siendo popular y es adecuado para proyectos que requieren un rendimiento más alto o mayor personalización.

4. Conclusión

- **GSON** y **Jackson** son las bibliotecas más populares, con **GSON** como la opción preferida debido a su velocidad y flexibilidad.
- **JSON-P** y **JSON-B** son adecuados para aplicaciones más complejas que requieren un procesamiento directo de JSON o una conversión sencilla entre objetos Java y JSON, respectivamente.
- **mJson** y **Boon** han sido descontinuadas y no se recomiendan para nuevos proyectos.