XML con Jackson

Contenido

[Consultar el archivo XML 1](#_Toc147216053)

[Añadir un nuevo producto 2](#_Toc147216054)

[Actualizar el precio de un producto. 2](#_Toc147216055)

[Eliminar un producto del cliente 3](#_Toc147216056)

[Convertir datos XML a JSON 4](#_Toc147216057)

[Conversión de XML (con un atributo: nombre) a JSON 4](#_Toc147216058)

[Dependencias. 5](#_Toc147216059)

[La clase JsonNode. 6](#_Toc147216060)

Archivo XML

<?xml version='1.0' encoding='UTF-8'?>

<Datos>

<cliente>

<nombre>Maria Lara</nombre>

<edad>30</edad>

<direccion>

<calle>Calle Ejemplo</calle>

<ciudad>Ciudad Ejemplo</ciudad>

</direccion>

</cliente>

<productos>

<productos>

<nombre>Producto 2</nombre>

<precio>9.99</precio>

</productos>

<productos>

<nombre>Producto 3</nombre>

<precio>49.99</precio>

</productos>

<productos>

<nombre>Producto 4</nombre>

<precio>49.99</precio>

</productos>

<productos>

<nombre>Producto 1</nombre>

<precio>49.99</precio>

</productos>

<productos>

<nombre>Producto 1</nombre>

<precio>49.99</precio>

</productos>

</productos>

</Datos>

# Consultar el archivo XML

import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import java.io.File;

import java.io.IOException;

import modelo.\*;

**public class VerNombreEdadProductos {**

public static void main(String[] args) {

try {

XmlMapper xmlMapper = new XmlMapper();

// Leer y mapear el archivo XML a una estructura de Jackson

File xmlFile = new File("./ficheros/datos.xml");

Datos datos = xmlMapper.readValue(xmlFile, Datos.class);

// Mostrar el nombre y la edad de los clientes, junto con los productos

Cliente cliente = datos.getCliente();

System.out.println("Nombre del cliente: " + cliente.getNombre());

System.out.println("Edad del cliente: " + cliente.getEdad());

System.out.println("Productos del cliente:");

for (Producto producto : datos.getProductos()) {

System.out.println(" Nombre del producto: " + producto.getNombre());

}

System.out.println("-------------------");

} catch (IOException e) {

e.printStackTrace();

}

}

}

# Añadir un nuevo producto

import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import com.fasterxml.jackson.dataformat.xml.ser.ToXmlGenerator;

import java.io.File;

import java.util.\*;

import modelo.\*;

public class AñadirProductoXML {

public static void main(String[] args) {

try {

// Cargar el archivo XML existente

File xmlFile = new File("./ficheros/datos.xml");

XmlMapper xmlMapper = new XmlMapper();

Datos datos = xmlMapper.readValue(xmlFile, Datos.class);

// Crear un nuevo producto

Producto nuevoProducto = new Producto("Producto 1", 49.99); // Cambiar por los valores deseados

List<Producto> productos = datos.getProductos();

List<Producto> productosList = new ArrayList<>();

productosList.add(nuevoProducto);

datos.setProductos(productos);

// Guardar los cambios en el archivo

xmlMapper.configure(ToXmlGenerator.Feature.WRITE\_XML\_DECLARATION, true);

xmlMapper.writeValue(xmlFile, datos);

System.out.println("Nuevo producto agregado exitosamente.");

} catch (Exception e) {

e.printStackTrace();

}

}

}

# Actualizar el precio de un producto.

import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import com.fasterxml.jackson.dataformat.xml.ser.ToXmlGenerator;

import java.io.File;

import java.util.ArrayList;

import java.util.List;

import modelo.Datos;

import modelo.Producto;

public class ActualizarPrecioProducto {

public static void main(String[] args) {

try {

// Cargar el archivo XML existente

File xmlFile = new File("./ficheros/datos.xml");

XmlMapper xmlMapper = new XmlMapper();

Datos datos = xmlMapper.readValue(xmlFile, Datos.class);

// Modificar el precio de un producto por nombre

String productoAModificar = "Producto 2"; // Cambiar por el nombre del producto a modificar

double nuevoPrecio = 9.99; // Cambiar por el nuevo precio

List<Producto> productos = datos.getProductos();

List<Producto> productosList = new ArrayList<>();

for (Producto producto : productosList) {

if (producto.getNombre().equals(productoAModificar)) {

producto.setPrecio(nuevoPrecio);

break;

}

}

// Guardar los cambios en el archivo

xmlMapper.configure(ToXmlGenerator.Feature.WRITE\_XML\_DECLARATION, true);

xmlMapper.writeValue(xmlFile, datos);

System.out.println("Precio de producto modificado exitosamente.");

} catch (Exception e) {

e.printStackTrace();

}

}

}

# Eliminar un producto del cliente

import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import com.fasterxml.jackson.dataformat.xml.ser.ToXmlGenerator;

import java.io.File;

import java.util.ArrayList;

import java.util.List;

import modelo.Cliente;

import modelo.Datos;

import modelo.Producto;

public class EliminarProducto {

public static void main(String[] args) {

try {

// Cargar el archivo XML existente

File xmlFile = new File("./ficheros/datos.xml");

XmlMapper xmlMapper = new XmlMapper();

Datos datos = xmlMapper.readValue(xmlFile, Datos.class);

// Eliminar un producto por nombre

String productoAEliminar = "Producto 1"; // Cambiar por el nombre del producto a eliminar

Cliente cliente = datos.getCliente();

List<Producto> productos = datos.getProductos();

List<Producto> productosList = new ArrayList<>();

for (Producto producto : productosList) {

if (producto.getNombre().equals(productoAEliminar)) {

productosList.remove(producto);

break;

}

}

datos.setProductos(productosList);

// Guardar los cambios en el archivo

xmlMapper.configure(ToXmlGenerator.Feature.WRITE\_XML\_DECLARATION, true);

xmlMapper.writeValue(xmlFile, datos);

System.out.println("Producto eliminado exitosamente.");

} catch (Exception e) {

e.printStackTrace();

}

}

}

# Convertir datos XML a JSON

import com.fasterxml.jackson.databind.JsonNode;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.SerializationFeature;

import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import java.io.File;

import java.io.IOException;

**public class ConvertirXMLtoJSON {**

public static void main(String[] args) {

try {

// Ruta de los archivos de entrada y salida

String archivoXML = "./datos.xml";

String archivoJSON = "./datosJSON.json";

// Crear un objeto Mapper para XML y JSON

XmlMapper xmlMapper = new XmlMapper();

ObjectMapper jsonMapper = new ObjectMapper();

jsonMapper.enable(SerializationFeature.INDENT\_OUTPUT); // Para formatear el JSON

// Leer el archivo XML y convertirlo a JsonNode

JsonNode jsonNode = xmlMapper.readTree(new File(archivoXML));

// Convertir JsonNode a formato JSON y escribirlo en un archivo

jsonMapper.writeValue(new File(archivoJSON), jsonNode);

System.out.println("Conversión completada: XML -> JSON");

// Convertir JsonNode a formato XML y escribirlo en la consola

xmlMapper.writeValue(System.out, jsonNode);

} catch (IOException e) {

e.printStackTrace();

}

}

}

# Conversión de XML (con un atributo: nombre) a JSON

Puedes realizar la conversión de XML a JSON y agregar el atributo nombre del cliente como un campo adicional en el JSON resultante de esta manera

<datos>

<cliente nombre="María">

<edad>30</edad>

<direccion>

<calle>Calle Ejemplo</calle>

<ciudad>Ciudad Ejemplo</ciudad>

</direccion>

</cliente>

<productos>

<producto>

<nombre>Producto 1</nombre>

<precio>19.99</precio>

</producto>

<producto>

<nombre>Producto 2</nombre>

<precio>29.99</precio>

</producto>

</productos>

</datos>

import com.fasterxml.jackson.databind.JsonNode;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.node.ObjectNode;

import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import java.io.File;

import java.io.IOException;

public class XmlToJsonConverter {

public static void main(String[] args) {

try {

// Leer datos XML usando XmlMapper

XmlMapper xmlMapper = new XmlMapper();

**JsonNode xmlData** = xmlMapper.readTree(new File("datos.xml"));

// Obtener el nombre del cliente y crear el objeto JSON

String clienteNombre = xmlData.get("cliente").get("@nombre").asText();

ObjectNode jsonData = new ObjectMapper().createObjectNode();

jsonData.put("nombre", clienteNombre);

// Agregar otros campos y elementos al objeto JSON

jsonData.set("edad", xmlData.get("cliente").get("edad"));

jsonData.set("direccion", xmlData.get("cliente").get("direccion"));

jsonData.set("productos", xmlData.get("productos"));

// Convertir el objeto JSON resultante a formato JSON

ObjectMapper jsonMapper = new ObjectMapper();

String json = jsonMapper.writerWithDefaultPrettyPrinter().writeValueAsString(jsonData);

System.out.println(json);

} catch (IOException e) {

e.printStackTrace();

}

}

}

Este código convierte el XML a un objeto JsonNode, obtiene el nombre del cliente como atributo, crea un objeto JSON y agrega los campos y elementos necesarios. Finalmente, convierte el objeto JSON a una cadena JSON formateada.

# Dependencias.

Aquí están las dependencias que necesitas agregar a tu archivo de configuración de Maven (pom.xml) para utilizar Jackson y su soporte para XML:

<dependencies>

<!-- Jackson Core -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.13.0</version>

</dependency>

<!-- Jackson Databind -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.13.0</version>

</dependency>

<!-- Jackson Dataformat XML -->

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

<version>2.13.0</version>

</dependency>

<!-- ... Otras dependencias de tu proyecto ... -->

</dependencies>

# La clase JsonNode.

**La clase JsonNode** es parte de la librería Jackson y se utiliza para representar nodos individuales en una estructura de datos JSON. Proporciona varios métodos para acceder, manipular y trabajar con los datos JSON de manera programática.

Algunos de los métodos más útiles de la clase JsonNode:

* get(String fieldName): Devuelve el nodo correspondiente al nombre del campo especificado en el JSON.
* get(int index): Devuelve el nodo correspondiente al índice del array especificado en el JSON.
* asText(): Devuelve el contenido del nodo como una cadena de texto.
* asInt(): Devuelve el contenido del nodo como un valor entero.
* asDouble(): Devuelve el contenido del nodo como un valor de punto flotante (doble).
* asBoolean(): Devuelve el contenido del nodo como un valor booleano.
* isObject(): Verifica si el nodo es un objeto JSON.
* isArray(): Verifica si el nodo es un array JSON.
* fieldNames(): Devuelve un iterador que recorre los nombres de campo en un objeto JSON.
* elements(): Devuelve un iterador que recorre los elementos de un array JSON.
* size(): Devuelve el número de elementos en un array JSON o el número de campos en un objeto JSON.
* getNodeType(): Devuelve el tipo de nodo, que puede ser OBJECT, ARRAY, STRING, NUMBER, BOOLEAN, NULL, entre otros.
* path(String fieldName): Devuelve el nodo correspondiente al nombre de campo especificado, similar a get(), pero retorna un nodo vacío si el campo no se encuentra.
* with(String fieldName): Devuelve un nodo mutable que representa el nodo con el nombre de campo especificado.
* isNull(): Verifica si el nodo es un valor nulo.
* get(String fieldName, JsonNode defaultValue): Devuelve el nodo correspondiente al nombre de campo especificado, pero si no se encuentra, retorna el valor predeterminado proporcionado.

Estos son solo algunos de los métodos más comunes de la clase JsonNode. Puedes consultar la documentación de Jackson para obtener una lista completa de los métodos disponibles y su descripción detallada: [Clase JsonNode - Documentación de Jackson](https://fasterxml.github.io/jackson-databind/javadoc/2.13/com/fasterxml/jackson/databind/JsonNode.html)

https://fasterxml.github.io/jackson-databind/javadoc/2.13/com/fasterxml/jackson/databind/JsonNode.html