# Traitement JSON en Java

Pour utiliser le package `com.fasterxml.jackson` dans un projet Java pour le traitement JSON, vous pouvez exploiter la bibliothèque Jackson, qui est largement utilisée pour convertir des objets Java en JSON et vice versa. Voici un guide étape par étape pour vous aider à démarrer, en supposant que vous travaillez sur un projet basé sur Maven. Ce guide se concentre sur le module `jackson-databind`, qui fournit des fonctionnalités de haut niveau pour la liaison de données JSON.

---

### 1. Ajouter la dépendance Jackson à votre projet

Pour utiliser le package `com.fasterxml.jackson`, vous devez inclure la bibliothèque Jackson dans votre projet. Si vous utilisez Maven, ajoutez la dépendance suivante à votre fichier `pom.xml` :

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.5</version> <!-- Remplacez par la dernière version -->
</dependency>
```

- **Note** : Vérifiez le Maven Central Repository pour la dernière version, car elle peut être plus récente que 2.12.5.
- Le module `jackson-databind` dépend de `jackson-core` et `jackson-annotations`, donc vous n'avez pas besoin de les ajouter séparément sauf si vous avez des exigences spécifiques.

Après avoir ajouté la dépendance, exécutez `mvn install` ou rafraîchissez votre projet dans votre IDE pour télécharger la bibliothèque.

---

### 2. Créer une instance `ObjectMapper`

La classe `ObjectMapper` du package `com.fasterxml.jackson.databind` est l'outil principal pour les opérations JSON. Elle est thread-safe et coûteuse en ressources à instancier, donc il est préférable de créer une seule instance réutilisable :

```
import com.fasterxml.jackson.databind.ObjectMapper;

public class JsonExample {
    private static final ObjectMapper mapper = new ObjectMapper();
}
```

Placez ceci dans une classe où vous effectuerez des opérations JSON.

---

### 3. Convertir un objet Java en JSON (Sérialisation)

Pour convertir un objet Java en une chaîne JSON, utilisez la méthode `writeValueAsString`. Voici un exemple :

**Définir une classe Java** Créez une classe avec les champs que vous souhaitez sérialiser. Assurez-vous qu'elle dispose de getters et de setters, car Jackson les utilise par défaut pour accéder aux champs privés :

```
public class MyClass {
    private String field1;
    private int field2;

    public MyClass(String field1, int field2) {
        this.field1 = field1;
        this.field2 = field2;
    }

    public String getField1() {
        return field1;
    }

    public void setField1(String field1) {
        this.field1 = field1;
    }

    public int getField2() {
        return field2;
    }

    public void setField2(int field2) {
        this.field2 = field2;
    }
}
```

**Sérialiser en JSON** Utilisez `ObjectMapper` pour convertir l'objet en JSON :

```
import com.fasterxml.jackson.databind.ObjectMapper;
```

```java
public class JsonExample {
    private static final ObjectMapper mapper = new ObjectMapper();

    public static void main(String[] args) throws Exception {
        MyClass obj = new MyClass("value1", 123);
        String json = mapper.writeValueAsString(obj);
        System.out.println(json);
    }
}
```

**Sortie** :

```
{"field1":"value1","field2":123}
```

---

### 4. Convertir JSON en objet Java (Désérialisation)

Pour convertir une chaîne JSON en objet Java, utilisez la méthode `readValue` :

```java
import com.fasterxml.jackson.databind.ObjectMapper;

public class JsonExample {
    private static final ObjectMapper mapper = new ObjectMapper();

    public static void main(String[] args) throws Exception {
        String json = "{\"field1\":\"value1\",\"field2\":123}";
        MyClass obj = mapper.readValue(json, MyClass.class);
        System.out.println(obj.getField1()); // Affiche "value1"
    }
}
```

- **Gestion des erreurs** : La méthode `readValue` lance une `JsonProcessingException` (une exception vérifiée) si le JSON est mal formé ou ne correspond pas à la structure de la classe. Gérez-la avec un bloc try-catch ou déclarez-la dans la signature de la méthode :

```java
try {
    MyClass obj = mapper.readValue(json, MyClass.class);
} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

---

### 5. Personnaliser le traitement JSON avec des annotations

Jackson fournit des annotations pour personnaliser la manière dont les champs sont sérialisés ou désérialisés. Ajoutez ces annotations du package `com.fasterxml.jackson.annotation` à votre classe :

**Renommer un champ** Utilisez `@JsonProperty` pour mapper un champ Java à un nom de champ JSON différent :

```
import com.fasterxml.jackson.annotation.JsonProperty;

public class MyClass {
    @JsonProperty("name")
    private String field1;
    private int field2;
    // Constructeur, getters, setters
}
```

**Sortie** :

```
{"name":"value1","field2":123}
```

**Ignorer un champ** Utilisez `@JsonIgnore` pour exclure un champ de la sérialisation :

```
import com.fasterxml.jackson.annotation.JsonIgnore;

public class MyClass {
    private String field1;
    @JsonIgnore
    private int field2;
    // Constructeur, getters, setters
}
```

**Sortie** :

```
{"field1":"value1"}
```

**Formater les dates** Utilisez `@JsonFormat` pour spécifier comment les dates sont sérialisées :

```
import com.fasterxml.jackson.annotation.JsonFormat;
import java.util.Date;
```

```java
public class MyClass {
    private String field1;
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd")
    private Date date;
    // Constructeur, getters, setters
}
```

**Sortie** (exemple) :

```
{"field1":"value1","date":"2023-10-25"}
```

---

### 6. Gérer des scénarios avancés

Voici quelques fonctionnalités supplémentaires que vous pourriez trouver utiles :

**Afficher JSON de manière lisible** Pour une sortie JSON lisible, utilisez `writerWithDefaultPrettyPrinter` :

```java
String prettyJson = mapper.writerWithDefaultPrettyPrinter().writeValueAsString(obj);
```

**Sortie** :

```
{
  "field1" : "value1",
  "field2" : 123
}
```

**Ignorer les propriétés inconnues** Si le JSON contient des champs non présents dans votre classe Java, configurez `ObjectMapper` pour les ignorer :

```java
mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
```

**Travailler avec des fichiers** Lire à partir de ou écrire dans un fichier :

```java
// Écrire dans un fichier
mapper.writeValue(new File("output.json"), obj);

// Lire à partir d'un fichier
MyClass obj = mapper.readValue(new File("input.json"), MyClass.class);
```

**Travailler avec des listes ou des génériques** Pour les collections, utilisez `TypeReference` pour préserver les informations de type générique :

```
import com.fasterxml.jackson.core.type.TypeReference;
import java.util.List;

List<MyClass> list = mapper.readValue(json, new TypeReference<List<MyClass>>() {});
```

---

### 7. Manipulation dynamique JSON avec `JsonNode`

Si vous ne souhaitez pas mapper JSON à une classe spécifique, utilisez `JsonNode` pour une approche basée sur un arbre :

```
import com.fasterxml.jackson.databind.JsonNode;

JsonNode node = mapper.readTree("{\"field1\":\"value1\",\"field2\":123}");
String value = node.get("field1").asText(); // Retourne "value1"
```

Vous pouvez également construire JSON de manière dynamique :

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

ObjectNode node = mapper.createObjectNode();
node.put("field1", "value1");
node.put("field2", 123);
String json = mapper.writeValueAsString(node);
```

---

### Résumé

Pour utiliser le package `com.fasterxml.jackson` : 1. Ajoutez la dépendance `jackson-databind` à votre `pom.xml`. 2. Créez une seule instance `ObjectMapper` pour la réutiliser. 3. Utilisez `writeValueAsString` pour convertir des objets Java en JSON et `readValue` pour convertir JSON en objets Java. 4. Personnalisez le comportement avec des annotations comme `@JsonProperty`, `@JsonIgnore` et `@JsonFormat`. 5. Gérez les exceptions (par exemple, `JsonProcessingException`) et configurez `ObjectMapper` pour des besoins spécifiques (par exemple, ignorer les propriétés inconnues). 6. Explorez des fonctionnalités avancées comme l'affichage lisible, l'E/S de fichiers ou `JsonNode` pour la manipulation dynamique de JSON.

Avec ces étapes, vous pouvez traiter efficacement le JSON dans votre application Java en utilisant la bibliothèque Jackson.