# Utilisation de NIO en Java

Me laissez-vous expliquer comment utiliser NIO (New Input/Output) en Java avec quelques exemples pratiques. Java NIO est une API d'E/S non bloquante qui offre une manière plus efficace de gérer les opérations d'E/S par rapport au package traditionnel java.io.

Voici un aperçu des composants clés et un exemple simple :

### Composants NIO Clés

1. **Buffers** : Conteneurs pour les données (par exemple, `ByteBuffer`).
2. **Canaux** : Flux pour lire/écrire des données (par exemple, `FileChannel`, `SocketChannel`).
3. **Sélecteurs** : Pour multiplexer plusieurs canaux en mode non bloquant.

### Exemple de Base : Lecture d'un Fichier avec NIO

Voici comment vous pouvez lire un fichier en utilisant `FileChannel` et `ByteBuffer` :

```
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class NIOFileReader {
    public static void main(String[] args) {
        try {
            // Ouvrir un canal de fichier
            FileChannel fileChannel = FileChannel.open(
                Paths.get("example.txt"),
                StandardOpenOption.READ
            );

            // Allouer un buffer
            ByteBuffer buffer = ByteBuffer.allocate(1024);

            // Lire les données dans le buffer
            while (fileChannel.read(buffer) > 0) {
                // Basculer le buffer pour préparer la lecture
                buffer.flip();

                // Traiter les données
```

```
                while (buffer.hasRemaining()) {
                    System.out.print((char) buffer.get());
                }

                // Effacer le buffer pour la prochaine lecture
                buffer.clear();
            }

            // Fermer le canal
            fileChannel.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

**Fonctionnement**

1. **FileChannel** : Ouvre une connexion au fichier.
2. **ByteBuffer** : Agit comme un stockage temporaire pour les données. Vous allouez sa taille (par exemple, 1024 octets).
3. **Opérations de Buffer** :
   - `flip()` : Prépare le buffer à être lu après avoir écrit des données dedans.
   - `clear()` : Réinitialise le buffer pour le prochain bloc de données.
4. **Lecture** : La méthode `read()` remplit le buffer, et vous extrayez les données octet par octet.

---

**Écriture dans un Fichier avec NIO**

Voici un exemple d'écriture de données dans un fichier :

```
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class NIOFileWriter {
    public static void main(String[] args) {
        try {
```

```
            // Ouvrir un canal de fichier pour l'écriture
            FileChannel fileChannel = FileChannel.open(
                Paths.get("output.txt"),
                StandardOpenOption.CREATE,
                StandardOpenOption.WRITE
            );

            // Données à écrire
            String data = "Hello, NIO!";
            ByteBuffer buffer = ByteBuffer.allocate(1024);

            // Mettre les données dans le buffer
            buffer.put(data.getBytes());

            // Basculer le buffer pour préparer l'écriture
            buffer.flip();

            // Écrire le buffer dans le fichier
            fileChannel.write(buffer);

            // Fermer le canal
            fileChannel.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

---

**Exemple Non-Bloquant avec Sélecteur**

Pour la programmation réseau (par exemple, un serveur), vous pouvez utiliser `Selector` avec `SocketChannel` :

```
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.net.InetSocketAddress;
```

```java
import java.util.Iterator;

public class NIOServer {
    public static void main(String[] args) throws Exception {
        // Ouvrir un sélecteur
        Selector selector = Selector.open();

        // Ouvrir un canal de socket serveur
        ServerSocketChannel serverChannel = ServerSocketChannel.open();
        serverChannel.bind(new InetSocketAddress(8080));
        serverChannel.configureBlocking(false);

        // Enregistrer le canal avec le sélecteur
        serverChannel.register(selector, SelectionKey.OP_ACCEPT);

        System.out.println("Serveur démarré sur le port 8080");

        while (true) {
            selector.select(); // Bloquer jusqu'à ce que des événements se produisent
            Iterator<SelectionKey> keys = selector.selectedKeys().iterator();

            while (keys.hasNext()) {
                SelectionKey key = keys.next();
                keys.remove();

                if (key.isAcceptable()) {
                    // Accepter une nouvelle connexion
                    SocketChannel client = serverChannel.accept();
                    client.configureBlocking(false);
                    client.register(selector, SelectionKey.OP_READ);
                    System.out.println("Client connecté");
                } else if (key.isReadable()) {
                    // Lire les données du client
                    SocketChannel client = (SocketChannel) key.channel();
                    ByteBuffer buffer = ByteBuffer.allocate(1024);
                    int bytesRead = client.read(buffer);
                    if (bytesRead == -1) {
                        client.close();
                    } else {
                        buffer.flip();
```

```
                    System.out.println("Reçu : " + new String(buffer.array(), 0, bytesRead));
                    buffer.clear();
                }
            }
        }
    }
}
```

**Conseils pour Utiliser NIO**

- **Gestion des Buffers** : Toujours `flip()` après avoir écrit dans un buffer et avant de lire, et `clear()` lors de la réutilisation.
- **Non-Bloquant** : Utilisez `configureBlocking(false)` et `Selector` pour des applications réseau évolutives.
- **Performance** : NIO est performant dans les scénarios avec de nombreuses connexions concurrentes ou des transferts de grandes quantités de données.

Faites-moi savoir si vous souhaitez un approfondissement sur une partie spécifique !