Usando NIO en Java

Déjame explicar cómo usar NIO (New Input/Output) en Java con algunos ejemplos prácticos. Java NIO es una API de E/S no bloqueante que proporciona una manera más eficiente de manejar operaciones de E/S en comparación con el paquete tradicional java.io.

Aquí tienes un desglose de los componentes clave y un ejemplo simple:

Componentes Clave de NIO

1. **Buffers**: Contenedores para datos (por ejemplo, `ByteBuffer`).
2. **Canales**: Flujos para leer/escibir datos (por ejemplo, `FileChannel`, `SocketChannel`).
3. **Selectores**: Para multiplexar múltiples canales en modo no bloqueante.

Ejemplo Básico: Lectura de un Archivo con NIO

Aquí tienes cómo puedes leer un archivo usando `FileChannel` y `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 {
            // Abre un canal de archivo
            FileChannel fileChannel = FileChannel.open(
                Paths.get("example.txt"),
                StandardOpenOption.READ
            );

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

            // Lee datos en el buffer
            while (fileChannel.read(buffer) > 0) {
                // Cambia el buffer para preparar la lectura
                buffer.flip();

                // Procesa los datos
```

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

                // Limpia el buffer para la próxima lectura
                buffer.clear();
            }

            // Cierra el canal
            fileChannel.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Cómo Funciona

1. **FileChannel**: Abre una conexión al archivo.
2. **ByteBuffer**: Actúa como un almacenamiento temporal para datos. Asignas su tamaño (por ejemplo, 1024 bytes).
3. **Operaciones del Buffer**:
   - `flip()`: Prepara el buffer para ser leído después de escribir datos en él.
   - `clear()`: Reinicia el buffer para el siguiente bloque de datos.
4. **Lectura**: El método `read()` llena el buffer, y extraes los datos byte a byte.

Escritura en un Archivo con NIO

Aquí tienes un ejemplo de escribir datos en un archivo:

```
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 {
```

```
            // Abre un canal de archivo para escribir
            FileChannel fileChannel = FileChannel.open(
                Paths.get("output.txt"),
                StandardOpenOption.CREATE,
                StandardOpenOption.WRITE
            );

            // Datos para escribir
            String data = "¡Hola, NIO!";
            ByteBuffer buffer = ByteBuffer.allocate(1024);

            // Coloca datos en el buffer
            buffer.put(data.getBytes());

            // Cambia el buffer para preparar la escritura
            buffer.flip();

            // Escribe el buffer en el archivo
            fileChannel.write(buffer);

            // Cierra el canal
            fileChannel.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Ejemplo No Bloqueante con Selector

Para la programación de red (por ejemplo, un servidor), puedes usar `Selector` con `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;
import java.util.Iterator;
```

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

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

        // Registra el canal con el selector
        serverChannel.register(selector, SelectionKey.OP_ACCEPT);

        System.out.println("Servidor iniciado en el puerto 8080");

        while (true) {
            selector.select(); // Bloquea hasta que ocurran eventos
            Iterator<SelectionKey> keys = selector.selectedKeys().iterator();

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

                if (key.isAcceptable()) {
                    // Acepta nueva conexión
                    SocketChannel client = serverChannel.accept();
                    client.configureBlocking(false);
                    client.register(selector, SelectionKey.OP_READ);
                    System.out.println("Cliente conectado");
                } else if (key.isReadable()) {
                    // Lee datos del cliente
                    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("Recibido: " + new String(buffer.array(), 0, bytesRead));
```

```
                    buffer.clear();
                }
            }
        }
    }
}
```

Consejos para Usar NIO

- **Gestión de Buffers**: Siempre `flip()` después de escribir en un buffer y antes de leer, y `clear()` al reutilizarlo.
- **No Bloqueante**: Usa `configureBlocking(false)` y `Selector` para aplicaciones de red escalables.
- **Rendimiento**: NIO brilla en escenarios con muchas conexiones concurrentes o transferencias de datos grandes.

Házmelo saber si te gustaría un análisis más profundo de alguna parte específica.