# 使用 NIO 於 Java

讓我解釋如何在 Java 中使用 NIO（新輸入/輸出）以及一些實際範例。Java NIO 是一個非阻塞 I/O API，提供了一種比傳統 java.io 套件更高效的處理 I/O 操作的方式。

以下是關鍵組件的分解以及一個簡單的範例：

### 關鍵 NIO 組件

1. **Buffers**：數據容器（例如 `ByteBuffer`）。
2. **Channels**：用於讀取/寫入數據的流（例如 `FileChannel`、`SocketChannel`）。
3. **Selectors**：用於在非阻塞模式下多路複用多個通道。

### 基本範例：使用 NIO 讀取文件

以下是如何使用 `FileChannel` 和 `ByteBuffer` 讀取文件的方法：

```
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 {
            // 打開文件通道
            FileChannel fileChannel = FileChannel.open(
                Paths.get("example.txt"),
                StandardOpenOption.READ
            );

            // 分配一個緩衝區
            ByteBuffer buffer = ByteBuffer.allocate(1024);

            // 將數據讀入緩衝區
            while (fileChannel.read(buffer) > 0) {
                // 翻轉緩衝區以準備讀取
                buffer.flip();

                // 處理數據
                while (buffer.hasRemaining()) {
```

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

            // 清除緩衝區以便下次讀取
            buffer.clear();
        }

        // 關閉通道
        fileChannel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

### 工作原理

1. **FileChannel**：打開到文件的連接。
2. **ByteBuffer**：作為數據的臨時存儲。您分配其大小（例如 1024 字節）。
3. **緩衝區操作**：
   - `flip()`：在將數據寫入緩衝區後準備讀取。
   - `clear()`：重置緩衝區以便下一個數據塊。
4. **讀取**：`read()` 方法填充緩衝區，您按字節提取數據。

---

### 使用 NIO 寫入文件

以下是將數據寫入文件的範例：

```
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 {
            // 打開用於寫入的文件通道
            FileChannel fileChannel = FileChannel.open(
```

```
                Paths.get("output.txt"),
                StandardOpenOption.CREATE,
                StandardOpenOption.WRITE
            );

            // 要寫入的數據
            String data = "Hello, NIO!";
            ByteBuffer buffer = ByteBuffer.allocate(1024);

            // 將數據放入緩衝區
            buffer.put(data.getBytes());

            // 翻轉緩衝區以準備寫入
            buffer.flip();

            // 將緩衝區寫入文件
            fileChannel.write(buffer);

            // 關閉通道
            fileChannel.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

---

**使用 Selector 的非阻塞範例**

對於網絡編程（例如伺服器），您可以使用 `Selector` 與 `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;

public class NIOServer {
```

```
public static void main(String[] args) throws Exception {
    // 打開選擇器
    Selector selector = Selector.open();

    // 打開伺服器套接字通道
    ServerSocketChannel serverChannel = ServerSocketChannel.open();
    serverChannel.bind(new InetSocketAddress(8080));
    serverChannel.configureBlocking(false);

    // 將通道與選擇器註冊
    serverChannel.register(selector, SelectionKey.OP_ACCEPT);

    System.out.println(" 伺服器在 8080 端口啟動");

    while (true) {
        selector.select(); // 阻塞直到事件發生
        Iterator<SelectionKey> keys = selector.selectedKeys().iterator();

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

            if (key.isAcceptable()) {
                // 接受新連接
                SocketChannel client = serverChannel.accept();
                client.configureBlocking(false);
                client.register(selector, SelectionKey.OP_READ);
                System.out.println(" 客戶端連接");
            } else if (key.isReadable()) {
                // 從客戶端讀取數據
                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(" 接收到: " + new String(buffer.array(), 0, bytesRead));
                    buffer.clear();
                }
```

```
                }
            }
        }
    }
}
```

### 使用 NIO 的技巧

- **緩衝區管理**：在寫入緩衝區後和讀取前總是 `flip()`，並在重用時 `clear()`。
- **非阻塞**：使用 `configureBlocking(false)` 和 `Selector` 以便於可擴展的網絡應用程序。
- **性能**：NIO 在有許多並發連接或大數據傳輸的情況下表現出色。

如果您想深入了解任何特定部分，請告訴我！