Java 物件序列化

我假設你指的是“java.io.Serializable”- 這是 Java 中用於物件序列化的關鍵介面。以下是如何使用它的實用指南：

什麼是 Serializable？

`java.io.Serializable` 是一個標記介面（沒有需要實現的方法），它告訴 Java 運行時，一個類別可以被序列化 - 轉換成字節流以保存到文件、通過網絡發送或存儲在數據庫中，然後再反序列化回物件。

基本步驟使用 Serializable

1. **實現介面** 在你的類別中添加 `implements Serializable`。這是最低要求。

```
import java.io.Serializable;

public class Person implements Serializable {
    private String name;
    private int age;

    // 建構子
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getters 和 setters
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}
```

2. **序列化 - 寫入物件** 使用 `ObjectOutputStream` 將物件寫入文件或流。

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
```

```
import java.io.IOException;

public class SerializeExample {
    public static void main(String[] args) {
        Person person = new Person("Alice", 25);

        try (FileOutputStream fileOut = new FileOutputStream("person.ser");
             ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
            out.writeObject(person);
            System.out.println(" 序列化數據已保存到 person.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3. **反序列化 - 讀取物件** 使用 `ObjectInputStream` 讀取物件。

```
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.IOException;

public class DeserializeExample {
    public static void main(String[] args) {
        try (FileInputStream fileIn = new FileInputStream("person.ser");
             ObjectInputStream in = new ObjectInputStream(fileIn)) {
            Person person = (Person) in.readObject();
            System.out.println(" 反序列化 Person: " + person);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

關鍵點要記住

- **serialVersionUID**: 為了在反序列化時確保兼容性，定義一個 `serialVersionUID`。如果不定義，Java 會自動生成一個，但它可能在不同的 JVM 或類別變更時有所不同。

```
private static final long serialVersionUID = 1L;
```

將這個添加到你的 `Person` 類別中。

- **暫時字段**: 使用 `transient` 修飾的字段不會被序列化。`java  private transient String temporaryData;`
- **不可序列化字段**: 如果一個字段的類別沒有實現 `Serializable`，你會得到一個 `NotSerializableException`。要麼讓那個類別實現 `Serializable`，要麼將字段標記為 `transient`。
- **繼承**: 如果超類別實現了 `Serializable`，子類別也會是可序列化的。如果沒有，超類別必須有一個無參數的建構子，以便反序列化能夠正常工作。

使用 transient 和 serialVersionUID 的範例

```
import java.io.Serializable;

public class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private transient String password; // 不會被序列化
    private int id;

    public Employee(String name, String password, int id) {
        this.name = name;
        this.password = password;
        this.id = id;
    }

    @Override
    public String toString() {
        return "Employee{name='" + name + "', password='" + password + "', id=" + id + "}";
    }
}
```

測試它

序列化和反序列化 `Employee`。你會注意到 `password` 在反序列化後是 `null`，因為它是 `transient`。

常見用例

- 將物件狀態保存到文件（例如，遊戲保存）。
- 通過網絡發送物件（例如，在 RMI 或分佈式系統中）。
- 緩存物件。

錯誤

- **安全性**: 如果不小心處理，序列化可能會暴露私有字段。對於敏感數據，考慮自定義 `writeObject` 和 `readObject` 方法。
- **性能**: 對於複雜物件，序列化可能會很慢。
- **版本控制**: 類別變更（例如，添加字段）可能會破壞反序列化，除非管理 `serialVersionUID`。

自定義序列化（可選）

如果你需要控制序列化，實現這些方法：

```
private void writeObject(ObjectOutputStream out) throws IOException {
    out.defaultWriteObject();
    // 自定義邏輯
}

private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
    in.defaultReadObject();
    // 自定義邏輯
}
```

讓我知道如果你想深入了解任何部分！