Introduction to Serialization and Deserialization

Nisal Madhuhansa

2301682030

GDSE 68

Contents

Introduction to Serialization and Deserialization

Serialization and deserialization are fundamental processes in software development, particularly in the context of data exchange and storage. Serialization is the act of converting an object or data structure into a format that can be stored or transmitted, such as a byte stream or a JSON string. Deserialization is the reverse process, where the serialized data is reconstructed back into its original object or data structure.

Benefits of Serialization and Deserialization

Data Persistence: Serialization allows you to save the state of an object or data structure to a file or a database, enabling data persistence across application sessions or system reboots.

Data Exchange: Serialized data can be easily transmitted between different systems, applications, or components, facilitating seamless data exchange and integration.

Efficient Data Storage: Serialized data often takes up less storage space compared to the original object or data structure, making it more efficient for storage and transmission.

Language Interoperability: Serialization and deserialization enable communication between applications written in different programming languages, as the serialized data can be understood and processed by different systems.

Mechanism of Serialization and Deserialization

The mechanism of serialization and deserialization typically involves the following steps:

Object Serialization: The object or data structure is converted into a stream of bytes or a textual representation (e.g., JSON, XML) that can be stored or transmitted.

Data Transmission: The serialized data is transmitted or stored in a file, database, or other storage medium.

Object Deserialization: The serialized data is read from the storage medium and reconstructed back into the original object or data structure.

Java code snippet that demonstrates the serialization and deserialization process:

java

Copy

artifact

id: serialization-example

name: Serialization and Deserialization Example

type: java

content: |-

```
import java.io.*;

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

    public Person(String name, int age) {
        this.name = name;
```

```
        this.age = age;
    }

    // Getters and setters
}

public class SerializationExample {
    public static void main(String[] args) {
        // Serialization
        Person person = new Person("John Doe", 35);
        try {
            FileOutputStream fileOut = new FileOutputStream("person.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(person);
            out.close();
            fileOut.close();
            System.out.println("Serialized data is saved in person.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Deserialization
        Person deserializedPerson;
        try {
            FileInputStream fileIn = new FileInputStream("person.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            deserializedPerson = (Person) in.readObject();
            in.close();
```

```
            fileIn.close();
            System.out.println("Deserialized Person: " + deserializedPerson.getName() + ", " +
deserializedPerson.getAge());
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

The code snippet demonstrates the serialization and deserialization of a Person object using Java's built-in serialization mechanism. The Person class implements the Serializable interface, which allows its instances to be serialized and deserialized.

In the serialization process, a Person object is created, and then it is written to a file named "person.ser" using the ObjectOutputStream. In the deserialization process, the serialized data is read from the file using the ObjectInputStream, and the Person object is reconstructed.