13. Сериализация/десериализация

Сериализация объекта — это способность объекта сохранять полную копию его и любых других объектов на которые он ссылается, используя поток вывода(например, во внешний файл). Таким образом, объект может быть воссоздан из сериализованной (сохраненной) копии немного позже, когда это потребуется.

Сериализация объектов, как новая возможность введенная в JDK 1.1, предоставляет функцию для преобразования групп или отдельных объектов, в поток битов или массив байтов, для хранения или передаче по сети. И как было сказано, данный поток битов или массив байтов, можно преобразовать обратно в объекты Java. Главным образом это происходит автоматически благодаря классам `ObjectInputStream` и `ObjectOutputStream`. Программист может решить реализовать эту функцию, путем реализации интерфейса `Serializable` при создании класса.

Чтобы полностью понять концепцию сериализации, надо иметь четкое понимание других двух концепций — **персистентности объектов** и **потоков**. Здесь мы немного расскажем о каждой из них, дабы вспомнить. Полное же пояснение касательно них, требовало бы по отдельной главе для каждой из этих концепций.

Потоки

Каждая программа должна записывать свои данные в место хранения или канал, и каждая программа должна считывать данные из канала или места хранения. В Java, эти каналы, куда программы записывают и откуда программы считывают данные, называются Потоками (Stream).

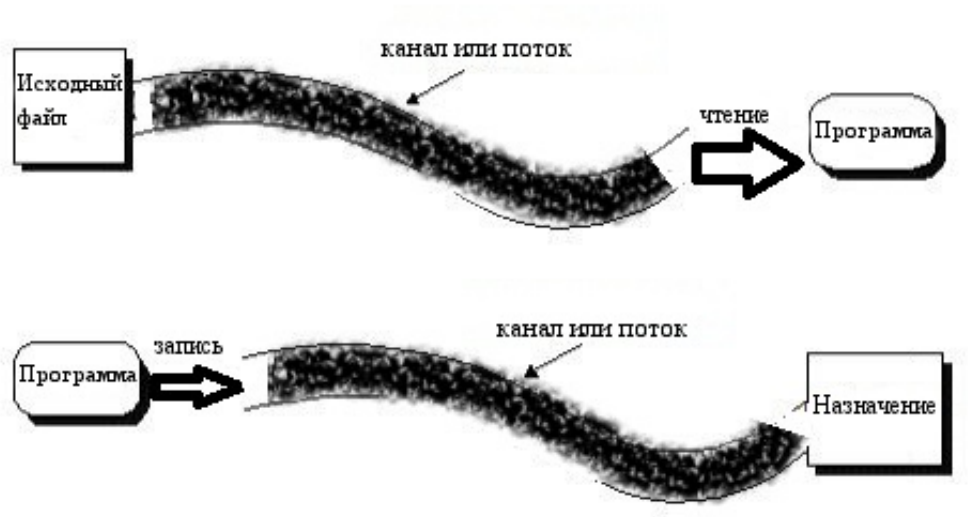

Рисунок 1. Графическое представление Потоков

Потоки в основном делятся на два типа:

- Байтовые классы-потоки именуемые *Streams

- Символьные классы-потоки именуемые *Reader и *Writer

Каждый поток записи данных, содержит набор методов записи. И каждый поток считывания данных, соответственно имеет подобный набор методов чтения. Как только поток создается, все эти методы должны быть вызваны.

Персистентность

Персистентность объекта — это способность обекта жить или по-другому — «пережить» выполнение программы. Это значит, что любой объект, который был создан во времени выполнения, уничтожается «мусорщиком» JVM, всякий раз, когда данный объект далее перестает использоваться. Но в случае реализации API персистентности, данные объекты не будут уничтожаться «мусорщиком» JVM, вместо чего им будет позволено «жить», что также дает возможность доступа к ним при следующем запуске приложения. Другими словами, персистенция означает существование времени жизни объекта, независимо от времени жизни приложения, которое запущено.

Один из способов реализации персистентности это хранение объектов где-нибудь во внешнем файле или в базе данных, а затем восстановление их в более позднее время, используя данные файлы или базу данных как источники. Здесь сериализация и вступает в игру. Любой неперсистентный объект существует так долго, как долго работает JVM.

Сериализованные объекты – это просто объекты, преобразованные в потоки, которые затем сохраняются во внешний файл или передаются через сеть для хранения и восстановления.

Реализация интерфейса Serializable

Любой класс должен реализовывать интерфейс `java.io.Serializable` для сериализации объектов этого класса. Интерфейс `Serializable` не имеет методов и только ~маркерует класс~, чтобы можно было ~идентифицировать его как сериализуемый~.

> Только ~поля объекта~ сериализованного класса могут быть сохранены.
>
> Методы или конструкторы не сохраняются, как части сериализованного потока.
>
> Если какой-либо объект действует как ссылка на другой объект, то поля этого объекта также сериализованны, если класс этого объекта реализует интерфейс `Serializable`.
>
> Другими словам, получаемый таким образом граф этого объекта, сериализуем полностью.
>
> Граф объекта включает дерево или структуру полей объекта и его подобъектов.

Два главных класса, которые помогают реализовать интерфейс `Seriliazable`:

- `ObjectInputStream`
- `ObjectOutputStream`

Листинг 1. Пример простого класса, чтобы показать сериализацию

```
import java.io.*;

public class RandomClass implements Serializable {
    // Генерация рандомного значения
    private static int r() {
        return (int)(Math.random() * 10);
    }
    private int data[];

    // Конструктор
    public RandomClass() {
        datafile = new int[r()];
        for (int i = 0; i < datafile.length; i++)
        datafile[i] = r();
     }

    public void printout() {
        System.out.println("This RandomClass has " + datafile.length + "
random integers");

        for (int i = 0; i < datafile.length; i++) {
            System.out.print(datafile[i] + ":");
            System.out.println();
        }
    }
}
```

В приведенном выше коде, создается класс, который является сериализуемым, т.к. «промаркерован» интерфейсом сериализации. Класс создает массив случайных целых чисел, когда создается его экземпляр.

Приведенный ниже код, показывает возможность записи объектов в поток, используя класс `ObjectOutputStream`. Программа имеет массив целых чисел, но для сериализации мы не должны перебирать ее внутренние объекты.

Интерфейс `Seriliazable` заботится об этом автоматически.

- ○ ■ -

Листинг 2. Простой пример сериализации объектов для вывода в файл

```
import java.io.*;
import java.util.*;

public class OutSerialize {
    public static void main (String args[]) throws IOException
    {
        RandomClass rc1 = new RandomClass();
        RandomClass rc2 = new RandomClass();

        // Создание цепи потоков с потоком вывода объекта в конце
        ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("objects.dat"));
        Date now = new Date(System.currentTimeMillis());

        out.writeObject(now);
        out.writeObject(rc1);
        out.writeObject(rc2);
        out.close();

        System.out.println("I have written:");
        System.out.println("A Date object: " + now);
        System.out.println("Two Group of randoms");

        rc1.printout();
        rc2.printout();
    }
}
```

Код ниже демонстрирует возможности класса `ObjectInputStream`, который считывает сериализованные данные с внешнего файла, в программу. Заметьте, что объекты считываются в том же порядке, в котором были записаны в файл.

- ○ ■ -

Листинг 3. Чтение сериализованных объектов или Десериализация

```
import java.io.*;
import java.util.*;

public class InSerialize {
    public static void main (String args[]) throws  IOException,
ClassNotFoundException {
        ObjectInputStream in =  new ObjectInputStream (new
FileInputStream("objects.dat"));
        Date d1 = (Date)in.readObject();

        RandomClass rc1 = (RandomClass)in.readObject();
        RandomClass rc2 = (RandomClass)in.readObject();

        System.out.println("I have read:");
        System.out.println("A Date object: " + d1);
        System.out.println("Two Group of randoms");

        rc1.printout();
        rc2.printout();
    }
}
```

Почти все классы Java могут быть сериализованны, включая классы AWT. Фрейм, который является окном, содержит набор графических компонентов. Если фрейм сериализован, механизм сериализации заботится об этом и сериализует все его компоненты и данные (позицию, содержание и т.д.). Некоторые объекты классов Java, не могут быть сериализованы, потому что содержат данные, что ссылаются на кратковременные ресурсы операционных систем. Например классы `java.io.FileInputStream` и `java.lang.Thread`. Если объект содержит ссылки на несериализуемые элементы, вся операция сериализации потерпит неудачу и будет выброшено исключение `NotSerializableException`. Если какой-либо объект ссылается на ссылку несериализованного объекта, то его можно сериализовать используя ключевое слово `transient`.

-

Листинг 4. Создание сериализуемых объектов используя ключевое слово `transient`

```
public class Sclass implements Serializable
{
    public transient Thread newThread; // помните, что поток (поток
параллельного исполнения) по умолчанию не сериализуемый класс
    private String studentID;
    private int sum;
}
```

Безопасность в Сериализации

Сериализация класса в Java, подразумевает передачу всех его данных во внешний файл или базу данных через поток. Мы можем ограничить данные, которые будут сериализованны, когда того пожелаем.

Есть два способа сделать это:

- Каждый параметр класса объявленный как `transient` и `static`, не сериализуются (по умолчанию все параметры класса сериализуются)
- Или каждый параметр класса, который мы хотим сериализовать, помечается тегом `Externalizable` (по умолчанию никакие параметры не сериализуются).

Поле данных не будет сериализовано с помощью `ObjectOutputStream`, когда оно будет вызвано для объекта, если это поле данных, данного объекта помечено как `transient`. Например – `private transient String password`.

С другой стороны, для явного объявления данных объекта как сериализуемых, мы должны промаркировать класс как `Externalizable`, и реализовать методы `writeExternal` и `readExternal` для записи и чтения данных этого объекта явно.

О том как работает Externalizable
Сериализация в Java / Хабрахабр