Serialización de objetos en Java

Descripción

Este artículo describe la serialización de objetos en Java, de forma aplicada, mediante ejemplos.

Introducción

La serialización de un objeto consiste en obtener una secuencia de bytes que represente el estado de dicho objeto. Esta secuencia puede utilizarse de varias maneras (puede enviarse a través de la red, guardarse en un fichero para su uso posterior, utilizarse para recomponer el objeto original, etc.).

Estado de un objeto

El estado de un objeto viene dado, básicamente, por el estado de sus campos. Así, serializar un objeto consiste, básicamente, en guardar el estado de sus campos. Si el objeto a serializar tiene campos que a su vez son objetos, habrá que serializarlos primero. Éste es un proceso recursivo que implica la serialización de todo un grafo (en realidad, un árbol) de objetos. Además, también se almacena información relativa a dicho árbol, para poder llevar a cabo la reconstrucción del objeto serializado.

En ocasiones puede interesar que un atributo concreto de un objeto no sea serializado. Esto se puede conseguir utilizando el modificador transient, que informa a la JVM de que no nos interesa mantener el valor de ese atributo para serializarlo o hacerlo persistente.

Ejemplo:

```
public class MiFecha
{
  protected int n;
  protected Date fecha;
  protected transient long s;
  . . .
}
```

En este ejemplo, los atributos n y fecha serán incluídos en la secuencia de bytes resultante de serializar un objeto de clase MiFecha. El atributo s no será incluído, por tener el modificador transient.

Objetos serializables. Interfaz Serializable

Un objeto serializable es un objeto que se puede convertir en una secuencia de bytes. Para que un objeto sea serializable, debe implementar la interfaz java.io.Serializable. Esta interfaz no define ningún método. Simplemente se usa para 'marcar' aquellas clases cuyas instancias pueden ser convertidas a secuencias de bytes (y posteriormente reconstruídas). Objetos tan comunes como String, Vector o ArrayList implementan Serializable, de modo que pueden ser serializados y reconstruídos más tarde.

Para serializar un objeto no hay más que declarar el objeto como serializable:

public class MiClase implements java.io.Serializable

El sistema de ejecución de Java se encarga de hacer la serialización de forma automática.

Ejemplos

Almacenamiento de objetos

Es posible utilizar los mecanismos de serialización disponibles para serializar un objeto guardándolo en un fichero y para realizar el proceso inverso, recuperándolo desde el fichero.

```
FileOutputStream fos = new FileOutputStream("fichero.bin");
FileInputStream fis = new FileInputStream("fichero.bin");
ObjectOutputStream out = new ObjectOutputStream(fos);
ObjectInputStream in = new ObjectInputStream(fis);

ClaseSerializable o1 = new ClaseSerializable();
ClaseSerializable o2 = new ClaseSerializable();

// Escribir el objeto en el fichero
out.writeObject(o1);
out.writeObject(o2);
. . .
// Leer el objeto del fichero (en el mismo orden !!)
o1 = (ClaseSerializable)in.readObject();
o2 = (ClaseSerializable)in.readObject();
```

Envío de objetos por la red

También es posible enviar un objeto serializado a través de la red. La diferencia consiste en que ahora se utilizan streams de distinto tipo.

```
Socket socket = new Socket(maquina, puerto);
OutputStream os = socket.getOutputStream();
InputStream is = socket.getInputStream();
ObjectOutputStream out = new ObjectOutputStream(os);
ObjectInputStream in = new ObjectInputStream(is);

PeticionSerializable ps = new PeticionSerializable();
RespuestaSerializable rs;

// Escribir una petición en el socket
out.writeObject(ps);
// Recibir del socket la respuesta
rs = (RespuestaSerializable)in.readObject();
```

Serialización en RMI

En RMI, la serialización se utiliza de forma casi transparente al usuario. Concretamente, se utiliza en el paso de parámetros y retorno de valores de las invocaciones a métodos de objetos remotos. Por ejemplo, cuando hacemos una invocación remota del tipo

```
retorno obj.metodo(param);
```

ocurre el siguiente proceso, de forma transparente al usuario:

1. (Local) El objeto param se serializa y se envía al objeto remoto como una secuencia de bytes
2. (Remoto) Se obtiene el objeto original a partir de la secuencia de bytes
3. (Remoto) Se ejecuta el método y se obtiene un valor de retorno
4. (Remoto) El valor de retorno se serializa y se envía como una secuencia de bytes
5. (Local) Se obtiene el retorno a partir de la secuencia de bytes

Para que esta invocación se lleve a cabo, es necesario que tanto los parámetros de las invocaciones remotas como los valores devueltos pertenezcan a clases serializables.

´

Serialización personalizada

En ocasiones puede interesar tomar el control sobre el proceso de serialización de una clase en concreto. Esto se puede hacer 'sobrecargando' los métodos writeObject y readObject de la clase cuya serialización se quiere controlar. En realidad, no se puede hablar de sobrecarga, puesto que estos métodos no están definidos en java.lang.Object. Este punto es un poco oscuro. Puede consultarse el API al respecto (método writeObject(Object) de ObjectOutputStream java.io.ObjectOutputStream y método readObject() de ObjectInputStream) y el JavaTutorial (Essential Java Classes -> Reading and Writing (but no 'rithmetic) -> Object Serialization -> Providing Object Serialization for Your Classes).

Para 'personalizar' la serialización de un objeto, basta añadir un método tal que:

```
private void writeObject (ObjectOutputStream stream)
             throws IOException
{
  stream.defaultWriteObject();
  . . .
}
```

Es necesario respetar exactamente tanto la signatura del método como la primera acción a realizar. A continuación pueden añadirse otras acciones que escriban en el stream dado.

También será necesario añadir un método para hacer el paso inverso:

```
private void readObject (ObjectInputStream stream)
             throws IOException
{
  stream.defaultReadObject();
  . . .
}
```

Aquí hay un ejemplo muy sencillo de uso de estos dos métodos. Con los println es posible comprobar que realmente se ejecutan cuando se produce la serialización y la reconstrucción de un objeto Prueba.

Interfaz Externalizable

Existe una interfaz en java.io llamada Externalizable que permite obtener un mayor control sobre el proceso de serialización y reconstrucción de nuestros objetos. Esta interfaz define dos métodos, writeExternal y readExternal, que se encargan de serializar y reconstruir un objeto, respectivamente.

La serialización mediante Externalizable requiere de un mayor cuidado. De forma automática, sólo se guarda información relativa a la identidad de la clase del objeto que se está serializando. No se guarda automáticamente ni su estado ni información relativa a sus superclases. Por ello, en la implementación de writeExternal hay que guardar explícitamente el estado de aquellos atributos que nos interesen, incluídos los heredados. A la hora de implementar writeExternal y readExternal, se ha de tener muy en cuenta la serialización de las clases superiores en el grafo de herencia y coordinar la implementación de estos métodos con la de los mismos métodos en clases superiores.

Emili Miedes es Ingeniero en Informática y trabaja de momento para el Instituto Tecnológico de Informática de la Universidad Politécnica de Valencia desarrollando en Java. Desarrollador opensource en sus ratos libres (buscar por *emiedes* en Sourceforge) anda buscando colaboración para acabar todo lo que empieza.
Para cualquier duda o tirón de orejas, e-mail a: emiedes@iti.upv.es