Persistenza e Serializzazione

JUG Sardegna – Cagliari, 30 Ottobre 2004

Pierangelo Caboni

Persistenza …

Spesso si ha la necessità che gli oggetti creati nel corso dell'esecuzione di un programma, non "muoiano" con la terminazione del programma, ma siano disponibili:

- nel corso di una successiva esecuzione dello stesso programma o di un altro, diverso, programma (*persistenza nel tempo*)
- in uno spazio di indirizzamento differente (*persistenza nello spazio*)

A tal scopo è evidente che occorre salvare tali oggetti su un supporto non volatile:

In altre parole: devono essere resi persistenti!

Valutare le esigenze …

- Quale supporto per la persistenza?
  - file? … che formato?
  - database relazionale (RDBMS)? … quale?
  - database ad oggetti (OODBMS)? … quale?

… non sempre ho libertà di scelta, magari la scelta è gia stata fatta da altri

- L'applicazione dovrà gestire uno o più di tali supporti?
- Dovranno essere gestite delle criticità particolari in termini di prestazioni o sicurezza?
- Il progetto che stiamo per affrontare si poggia su un'analisi solida o prevediamo che sarà facilmente suscettibile di modifiche?

… in ogni caso è bene

- accessibilità dei dati a più utenti "contemporaneamente"

(si pensi ad una qualunque applicazione realizzata per il web …)

- disponibilità di un certo controllo sulle transazioni

(i dati devono, devono rispettare dei vincoli d'integrità che ne garantiscono una coerenza complessiva)

- trasparenza, il codice dovrebbe essere privo di riferimenti al particolare supporto di persistenza

(a questo scopo è opportuno confinare il codice di acceso ai dati in un opportuno layer –DAO- o, meglio ancora, poterlo gestire con dei file di configurazione –xml-)

Come implementare la persistenza

Una volta individuate le reali esigenze e disponibilità, posso scegliere il meccanismo di persistenza o framework più adatto …

- Serializzazione
- JDBC
- XMI
- iBatis
- JDO
- Hybernate
- EJB

…

Serializzazione …

- Un meccanismo per la trasformazione automatica di oggetti (di qualunque complessità) in sequenze di byte
- Sequenze manipolabili con gli stream del package java.io:
  - per scrivere/leggere su/da file lo stato di un oggetto (FileOutputStream/FileInputStream - persistenza)
  - per scrivere/leggere su/da una connessione di rete (RMI)
  - per implementare un metodo alternativo di clonazione degli oggetti (ByteArrayOutputStream/ByteArrayInputStream)

Una classe è serializzabile se …

```
public class Film implements Serializable {
  private String titolo;
  private String regista;
  private int anno;

  public Film(String titolo, String regista, int anno) {
    this.titolo = titolo;
    this.regista = regista;
    this.anno = anno;  }

  public String toString() {
    String film = "[Film]\n";
    film += "Titolo:   " + this.titolo + "\n";
    film += "Regista:  " + this.regista + "\n";
    film += "Anno:     " + this.anno;
    return film;  } … }
```

condizione di seriabilizzalità

Serializable: un'interfaccia "anomala"

- Priva di metodi: informa il compilatore che la classe che la implementa è serializzabile (interfaccia *Marker*)

- Eccezione, a runtime, NotSerializableException, qualora si tenti di serializzare un oggetto di classe che non implementa l'interfaccia Serializable

- Le classi che estendono una classe serializzalibile sono anch'esse serializzabile, senza necessità di dichiararlo: un aspetto da tenere in considerazione in contesti che richiedono attenzione per la sicurezza

Il processo di *serializzazione* …

```
public class Serializza {
  public static void main(String[] args) throws IOException {

    Film film = new Film("Taxi Driver", "Martin Scorsese", 1976);

    // apertura dello stream di output (supporto di persistenza)
    OutputStream fos = new FileOutputStream("D:\\Film.ser");

    // apertura dello stream di serializzazione
    ObjectOutputStream oos = new ObjectOutputStream(fos);

    // scrittura su file della sequenza di byte
    // rappresentativa dello stato dell'oggetto
    oos.writeObject(film);
    oos.close();
    fos.close();
  }
}
```

… e quello di *deserializzazione*

```
public class Deserializza {
  public static void main(String[] args)
  throws IOException, ClassNotFoundException {
    // apertura stream di input
    InputStream fis = new FileInputStream("D:\\Film.ser");
    // apertura stream di deserializzazione
    ObjectInputStream ois = new ObjectInputStream(fis);
    // lettura e assegnazione dell'oggetto
    Film film = (Film)ois.readObject();
    ois.close();
    fis.close();
    System.out.println(film.toString());
  }
}
```

cast esplicito

Output

```
[Film]
Titolo:   Taxi Driver
Regista:  Martin Scorsese
Anno:     1976
```

Un catalogo di film …

… serializzo …

```
…
Film[] films = new Film[3];
films[0] = new Film("Taxi Driver", "Martin Scorsese", 1976);
films[1] = new Film("Rain Man", "Barry Lavinson", 1988);
films[2] = new Film("Balla coi Lupi", "Kevin Kostner", 1990);
…
oos.writeObject(films);
…
```

… deserializzo …

```
…
Film[] films = (Film[])ois.readObject();
for(int i=0; i<films.length; i++) {
  System.out.println(films[i].toString());
}
```

… nessuna variazione sul codice non riportato

… correttamente ripristinato

Output

```
[Film]
Titolo:    Taxi Driver
Regista:   Martin Scorsese
Anno:      1976

[Film]
Titolo:    Rain Man
Regista:   Barry Lavinson
Anno:      1988

[Film]
Titolo:    Balla coi Lupi
Regista:   Kevin Kostner
Anno:      1990
```

Si osservi che gli elementi dell'array vengono ripristinati nello stesso ordine in cui sono stati inseriti.
Ciò vale in generale: se avessimo usato un ArrayList, e vi avessimo inserito oggetti di tipo diverso, avremmo dovuto prestare attenzione all'ordine dei cast in fase di deserializzalione …

Copia superficiale e copia in profondità

Il meccanismo di serializzazione permette di salvare lo stato dell'oggetto nel suo complesso.

Ciò significa che:

- se in tale oggetto è presente un riferimento ad un altro oggetto viene salvato anche lo stato di tale oggetto (*copia in profondità*) e non semplicemente il riferimento ad esso (*copia superficiale*)

… motivo per cui "la rete di oggetti" deve essere costituita da istanze derivanti da classi serializzabili

- nel file binario, la rete di oggetti viene riprodotta esattamente con la stessa configurazione esistente in memoria prima della serializzazione

… gli indirizzi di memoria, privi di significati fuori dal contesto di esecuzione del programma, sono sostituiti da numeri seriali … motivo per cui si è adottato il termine serializzazione

Verifichiamo la copia in profondità …

… definiamo una classe Regista da usarsi, nella classe Film, al posto dell'attuale campo regista di tipo String.

```
public class Regista implements Serializable {
  private String nome;
  private String cognome;
  private String nazionalita;
  public Regista(String nome, String cognome, String nazionalita) {
    this.nome = nome;
    this.cognome = cognome;
    this.nazionalita = nazionalita;
  }
  public String toString() {
    return this.nome +" " +this.cognome +"(" + this.nazionalita +")";
  } … }
```

Adattiamo la classe Film …

```
public class Film implements Serializable {
  private String titolo;
  private Regista regista;
  private int anno;
  public Film(String titolo, Regista regista, int anno) {…}
  public String toString() {
    …
    film += "Regista:  " + this.regista + "\n";
  } … }
```

… a run time verrà invocato il metodo toString() definito nella classe Regista

… la verifica

… serializzo …

```
Regista regista = new Regista("Martin", "Scorsese", "USA");
Film film = new Film("Taxi Driver", regista, 1976);
…
try { oos.writeObject(film); }
catch (NotSerializableException ex) {
  System.out.println("Non serializzabile: " + ex.getMessage()); }
finally { /* chiusura stream */ }
```

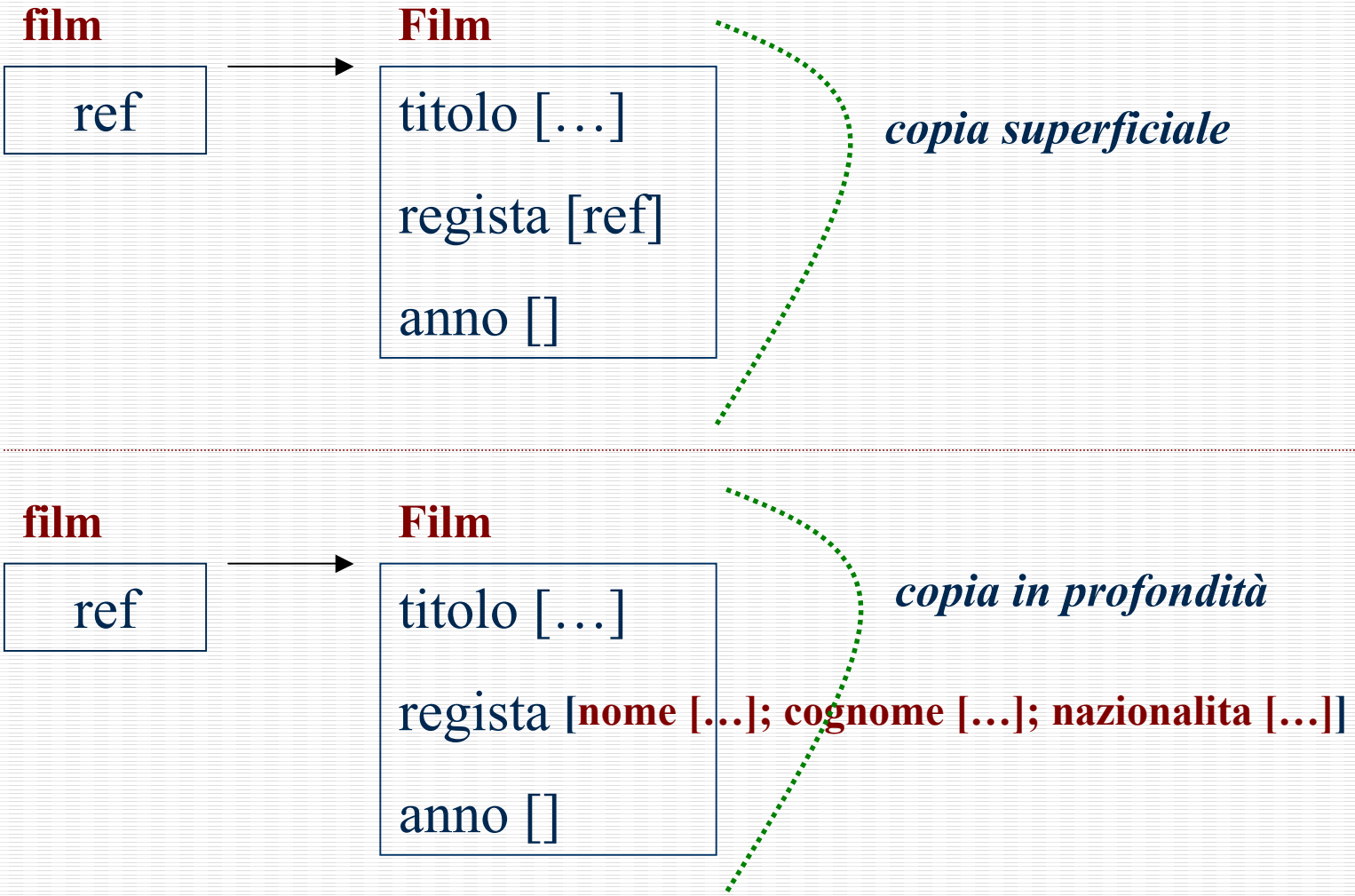
… deserializzo …

```
// nessuna modifica al codice
```

… un modo per individuare, nella rete di oggetti, quelli di classe non serializzabile

```
Output
[Film]
Titolo:   Taxi Driver
Regista:  Martin Scorsese (USA)
Anno:     1976
```

Un diagramma esemplificativo



Problemi di sicurezza …

Gli specificatori d'accesso sono ininfluenti rispetto alla serializzazione: un campo di istanza *private* viene comunque serializzato!

… ma un campo *password* sarebbe bene che non finisse su un file!!!

E' possibile controllare la serializzazione? SI

- usando lo specificatore *transient* per i campi che non si vuole vengano serializzati
  - `private transient String password;`
- implementando Externalizable piuttosto che Serializable
  - `public void readExternal(ObjectInput in)`
    `throws IOException, ClassNotFoundException;`
  - `public void writeExternal (ObjectOutput out)`
    `throws IOException;`

Spunti per approfondimenti

- Gestione delle versioni: cosa succede se modifico una classe che sia già stata oggetto di serializzazione?

- Quando serializzo due oggetti che contengono un riferimento ad uno stesso oggetto (ad esempio due film dello stesso regista), come viene realizzata la copia in profondità?

- Remote Method Invocation (RMI)

- Clonazione mediante serializzazione

Conclusioni

La Serializzazione

- e' semplice da implementare;
- e' intrinsecamente Object Oriented !!!

ma …

- non dispone di supporto per transazioni;
- non consente operazioni di recupero selettivo dei dati (*se non successivamente alla deserializzazione*)
- la gestione della sicurezza e delle versioni risulta essere onerosa.

Un suo impiego come meccanismo di persistenza è da prendersi in considerazione per progetti che non presentino particolari criticità in termini di sicurezza e non gestiscano notevoli quantità di dati.

Riferimenti

- Problematiche sulla persistenza

  Better, Faster, Lighter Java

  Tate – Gethtland, O'Really

- Serializzazione

  Thinking in Java

  B. Eckel

  Core Java 2

  Horstmann – Cornell, Prentice-Hall