Università degli Studi di Padova

Programmazione Concorrente e Distribuita

A.A. 2014/2015

Risolutore di Puzzle - Parte 3

Autore:

GIACOMO CUSINATO
Mat. 1054137

29 gennaio 2015

Indice

1 Ambiente di sviluppo e file di progetto

Il progetto è stato realizzato in ambiente Mac OS (versione Yosemite 10.10) sia per la parte relativa al codice Java, sia per quella relativa alla documentazione. È stato inoltre testato con successo nella macchine di laboratorio con i dovuti comandi per impostare Java 7 come versione principale del sistema. Per la stesura della relazione, invece, è stato utilizzato il linguaggio LaTeX tramite la ditribuzione MacTex. Come richiesto da specifica, oltre al codice sorgente sono stati inseriti i seguenti file:

- `Makefile`: si occupa di compilare i sorgenti Java
- `puzzlesolverserver.sh`: script bash che si occupa di effettuare il `make` del progetto ed avviare il main dell'applicativo server. Accetta come parametro il nome del server da passare al programma Java.
- `puzzlesolverclient.sh`: script bash che si occupa di effettuare il make del progetto, se necessario, ed avviare il main dell'applicativo client. Accetta tre parametri nel seguente ordine: path del file di input, path del file di output e nome del server.

Gli script sono stati resi eseguibili tramite il comando bash `"chmod 754 nomefile"` e possono essere lanciati semplicemente col comando `"./nomescript.sh args"` dalla cartella in cui è presente il file. Di seguito gli step per lanciare il programma in modo corretto:

1. posizionarsi nella cartella server e lanciare il comando `rmiregistry &`, in caso tale comando fosse già stato eseguito, il sistema scatenerà un'eccezione
2. spostarsi nella root del progetto ed avviare il server tramite lo script `puzzlesolverserver.sh` che accetta in input il nome dell'host come parametro (il nome dovrà essere quello di un server esistente, come `localhost`).
3. da una seconda shell, avviare il client tramite lo script `puzzlesolverclient.sh` che accetta come parametri il file di input, il file di output ed il nome del server (`localhost`).

2 Differenze con la versione precedente

Come da specifica, il progetto è stato suddiviso in modo tale da simulare l'interazione tra un client ed un server tramite tecnologia RMI. Tutta la logica e le classi relative ala risoluzione del puzzle sono state spostate nella parte server, in quanto il client si occuperà esclusivamente delle operazioni input/output ottenendo il risultato finale tramite una chiamata al metodo `reoder(String inputContent)` definito nell'oggetto remoto.

2.1 Classi lato server

Di seguito le classi che definiscono il modulo server del progetto:

- `Block`: rappresenta una tessera del puzzle. Rimane invariata rispetto alle precedenti consegne ma fa ora parte del modulo server.
- `PuzzleThread`: La classe PuzzleThread definisce un un oggetto la cui istanza può essere lanciata dal programma in un thread a supporto della risoluzione dell'algorimo. Anche questa classe rimane invarita rispetto alla precedente consegna ed è stata inserita nel modulo server.
- `PuzzleSolverServer`: è la classe principale del modulo server. Contiene la logica dell'algoritmo di risoluzione del puzzle (rimasta invariata) ed espone il metodo `reoder(String inputContent)` che può essere invocato da una JVM remota e restituisce la soluzione prodotta. Il metodo `main` crea un'istanza di questa classe e ne registra il riferimento nel registro RMI.

2.2 Classi lato client

Di seguito le classi che definiscono il modulo client del progetto:

- `IOHelper`: si occupa di effettura la lettura e la scrittura di stringhe su determinati file. Questa classe è rimasta invariata rispetto alle consegne precedenti.

- `PuzzleSolverClient`: è la casse principale del modulo client. Oltre a sfruttare i metodi esposti dalla classe `IOHelper`, si occupa di ottenere il riferimento all'oggetto remoto e chiamare il metodo `reoder(String inputContent)` così da ottenere la soluzione del puzzle tramite il server.

3 Implementazione del sistema RMI

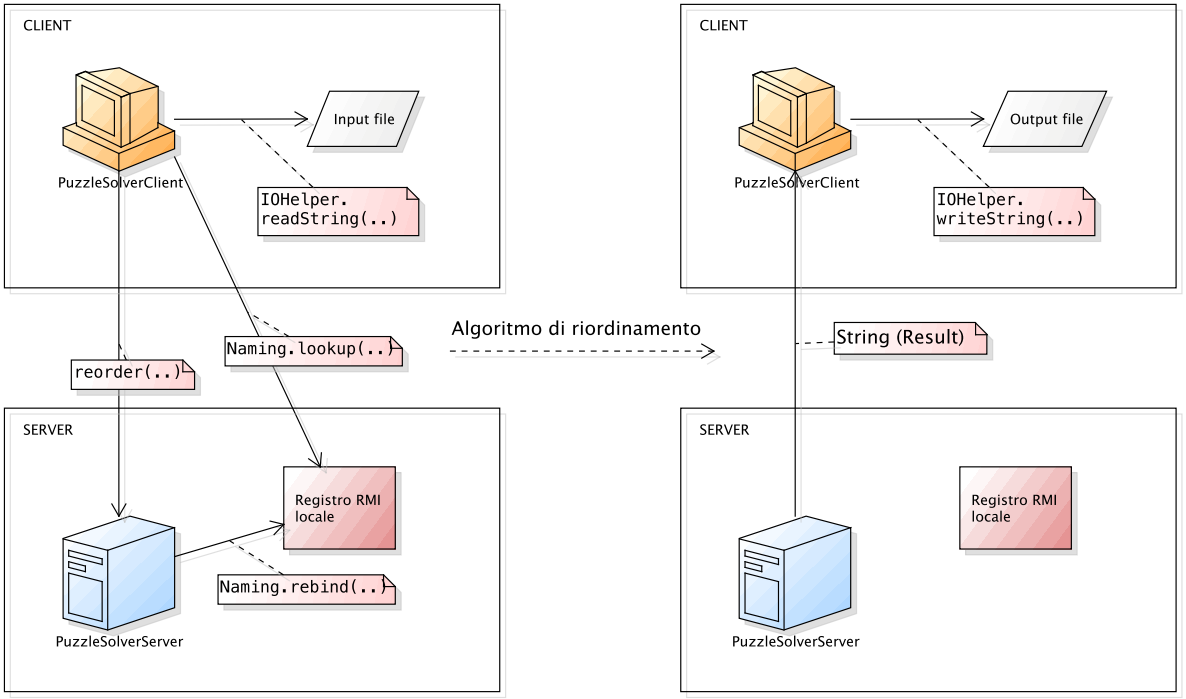

Figura 1: Vista generale implementazione RMI in PuzzleSolver

3.1 L'intefaccia ISolver

L'interfaccia `ISolver` estende l'interfaccia `Remote` che fa da semplice marcatore di tipo per l'oggetto remoto. `ISolver` contiene solamente la definizione del metodo remoto `reoder(String inputPath)`, che sarà implementato dalla classe `PuzzleSolverServer` con l'algoritmo di risoluzione. Tale metodo, può provocare un'eccezione di tipo `RemoteException` a causa di eventiuali problemi di connessione o comunicazione col server. L'interfaccia è presente sia nel modulo server che

nel modulo client, quest'ultimo, infatti, necessita esclusivamente della definizione del meteodo remoto da invocare visto che non sarà la JVM che risiede nel client ad eseguirlo.

3.2 La classe PuzzleSolverServer

L'instanza di questa classe rappresenta l'oggetto remoto il cui riferimento viene reso disponibile al client. Il `main` della classe, infatti, crea un oggetto di tipo `PuzzleSolverServer` e tramite il medoto statico `rebind(String uri, Remote obj)` della classe `Naming` registra l'instanza dell'oggetto creato nel registro RMI locale. La stringa `uri`, contenente il nome del server e l'identificativo associato all'oggetto, sarà utilizzata anche dal client per ottenere il riferimento a tale oggetto. La classe `PuzzleSolverServer` espone il metodo remoto `reorder(String inputPath)` che può essere invocato da una JVM remota ed esegue l'algoritmo di risoluzione del puzzle. Il metodo è stato dichiarato tramite la keyword `synchronized` in modo che il server possa gestire più client in attesa mentre esegue l'algoritmo per il client attivo. Il metodo `reset()`, inoltre, assicura che il valore di tutti i campi dati venga risportato a quello originale per il corretto funzionamento del metodo `reoder(String inputPath)` invocato dal client sucessivo.

3.3 La classe PuzzleSolverClient

La classe `PuzzleSolverClient`, una volta letto il contenuto dle file di input, si occupa di ottenere un riferimento all'oggetto instanziato sul server. Questo è possibile tramite il metodo statico `lookup(String uri)` della classe `Naming` che interroga il registro RMI remoto e ottiene il riferimento all'oggetto grazie all'`uri` identificativo associato. Inoltre il client, che contiene a sua volta la definizione dell'intefaccia `ISolver`, può ottenere il riferimento remoto effettuando un downcast proprio ad un oggetto di tipo `ISolver`, su cui è possibile invocare il metodo remoto `reorder(String inputPath)`.

3.4 Robustezza del sistema

Per risolvere un eventuale errore di connessione o comunicazione tra server e client, il metodo remoto `solver(String inputPath)` invocato dalla classe `PuzzleSolverClient`

è stato racchiuso in un costrutto try-catch-finally. In particolare, nel blocco finally, è stata inserita nuovamente la chiamata al metodo remoto, anch'esso con un'opportuna gestione delle eccezioni. In questo modo, in caso venga scatenata un'eccezione lato server o client, questa sarà incapsulata nel blocco catch e grazie al costrutto finally, verrà ritentata la chiamata al metodo remoto.