Università degli Studi di Bologna
Scuola di Ingegneria

Corso di
Reti di Calcolatori T

Esercitazione 6 (proposta)
Java RMI

Antonio Corradi, Luca Foschini
Michele Solimando, Giuseppe Martuscelli, Marco Torello
Anno Accademico 2020/2021

SPECIFICA

Utilizzando RMI sviluppare **un'applicazione C/S** che consenta di effettuare **alcune operazioni remote su file testo** per ottenere servizi da un server:

- in un file testo, **contare le righe che contengono un numero di parole superiore** ad **un intero** espresso dal cliente
- **eliminare una riga da un file remoto**. Il cliente invia al server il **nome del file** e il **numero di riga** e ottiene in risposta **l'esito dell'operazione**

I clienti siano sempre filtri ciclici e condizionati a lavorare fino ad esaurire tutti gli input dall'utente

il server deve comportarsi da processo demone sempre presente

METODI REMOTI (1/2)

Il progetto RMI si basa su un'**interfaccia remotizzabile** (**RemOp,** contenuta nel file RemOp.java)

in cui vengono definiti i **metodi invocabili in remoto dal client**

`conta_righe` e **`elimina_riga`**

In dettaglio, il metodo **`conta_righe`**:

- accetta come parametro il **nome di un file remoto** ed **un intero**, passati entrambi come parametri
- restituisce il **numero delle righe** che contengono un numero di parole maggiore dell'intero inviato; in caso di errore, solleva un'**eccezione remota**: ad esempio, se il file non è presente nel sistema o non è un file testo

Al solito, una parola è definita come una **sequenza di caratteri di lunghezza qualunque e delimitata da separatori** (consideriamo solo inizio linea / fine linea e spazi; pensare anche a come fare per averne altri …)

METODI REMOTI (2/2)

Il progetto RMI si basa su un'**interfaccia remotizzabile** (**RemOp,** contenuta nel file RemOp.java) in cui vengono definiti i **metodi invocabili in remoto dal client.**

Il metodo **`elimina_riga`**:

- accetta come parametri d'ingresso il **nome di un file remoto** ed **un intero**;
- se il file esiste e se ha un numero di righe almeno pari all'intero inviato dal cliente, restituisce **l'esito dell'operazione**, ovvero il nome del file modificato e un intero corrispondente al numero di righe presenti nel file modificato

  In caso di errore, solleva un'**eccezione remota**: ad esempio, se il file non è presente nel sistema, se non è un file testo o se ha meno righe di quella della quale se ne richiede l'eliminazione.

CLASSI IN GIOCO

Si progettino le classi:

- **ServerImpl** (contenuta nel file ServerImpl.java), che implementa i metodi del server invocabili in remoto e presenta l'interfaccia di invocazione:

```
ServerImpl [registryPort]
```

- **Client** (contenuta nel file Client.java), che realizza l'interazione con l'utente proponendo ciclicamente i servizi che utilizzano i due metodi remoti, e stampa a video i risultati, fino alla fine del file di input da tastiera. Il Client presenta l'interfaccia di invocazione:

```
Client RegistryHost [registryPort]
```

Note per recupero riferimento Server

Usuale **workflow dei tre elementi**

Il **Registry** deve essere in esecuzione su un host concordato e in ascolto alla porta eventualmente specificata

Il **Server** (istanza della classe relativa) deve registrare il riferimento remoto sul registry alla locazione corretta

Il **Client** (istanza della classe relativa) deve recuperare dal registry il riferimento all'oggetto remoto, ServerImpl, di cui deve invocare i metodi

PROPOSTA DI ESTENSIONE: TRASFERIMENTO DI UN DIRETTORIO

Si vuole sviluppare **un'applicazione C/S basata su RMI** e su socket con connessione per il trasferimento di tutti i file di un direttorio remoto dal **server al client** (**multiple get**)

In particolare, si vogliono realizzare due modalità di trasferimento, la prima con **client attivo** (**il client effettua la connect**), la seconda con **server attivo** (**il server effettua la connect**). Si dovranno realizzare un **client** e un **server**; l'utente, per ogni trasferimento, decide quale delle due modalità utilizzare

Per entrambe le modalità, si prevede **un'interazione iniziale sincrona (realizzata con una richiesta RMI sull'oggetto remoto server)** per trasferire la **lista dei file** da inviare e **l'endpoint** (host e porta) **di ascolto**; quindi, una seconda fase di effettivo **trasferimento dei file** dal **server al client** (**realizzata con socket connesse**)

TRASFERIMENTO PIÙ DIRETTORI: CLIENT ATTIVO

Il metodo remoto accetta come argomento di ingresso il **nome del direttorio** e restituisce una struttura dati con l'**endpoint di ascolto del server** e la **lista con i nomi e la lunghezza di tutti i file** da trasferire

Il **Client** richiede ciclicamente all'utente **il nome del direttorio** da trasferire ed **effettua la chiamata RMI e riceve l'endpoint di ascolto**, quindi **stabilisce una connessione con socket stream** con il server remoto e riceve i file salvandoli sul direttorio locale

Il **Server** implementa **il metodo RMI richiesto** ed è realizzato come **server concorrente** e **parallelo**. Per ogni nuova richiesta ricevuta il processo padre, dopo **aver accettato la richiesta RMI**, attiva un processo figlio a cui affida la **creazione della socket di ascolto** e il completamento del servizio richiesto; quindi il padre restituisce **la lista dei file** e **il proprio endpoint**

TRASFERIMENTO PIÙ DIRETTORI: SERVER ATTIVO

Il metodo remoto accetta come argomento di ingresso il **nome del direttorio** e l'**endpoint di ascolto del client** e restituisce la **lista con i nomi e la lunghezza di tutti i file** da trasferire

Il **Client** richiede ciclicamente all'utente **il nome del direttorio** da trasferire, **crea la socket di ascolto**, **poi effettua la chiamata RMI** e riceve la **lista dei file da trasferire**; infine, **effettua la accept** e attiva i trasferimenti di file necessari sulla connessione

Il **Server** implementa il metodo RMI richiesto ed è realizzato come **server concorrente** e **parallelo**; per ogni nuova richiesta ricevuta il processo padre, dopo aver **accettato la richiesta RMI**, attiva un processo figlio a cui affida la **creazione della socket** su cui eseguire la **connect** e il completamento del servizio richiesto; quindi **restituisce la lista dei file**