Appunti di Laboratorio di Reti

Simone Ianniciello

A.A. 2020/2021

Contents

Chapter 1

Threads

1.1 L'interfaccia Runnable

Permette la creazione di task eseguibili da uno o piu' threads. Contiene solamente la segnatura del metodo `void run()`. Il frammento di codice contenuto all'interno del metodo `run()` verrà eseguito richiamando il metodo `start()` di un thread associato.

1.2 Daemon Threads

Sono thread a bassa priorità, utili per servizi che devono essere eseguiti finché il programma e' in esecuzione. Non appena tutti i threads non demoni del programma sono terminati, la JVM forza la terminazione dei thread demoni.

1.3 Metodi

`t.join()` Blocca l'esecuzione del thread chiamante finché il thread `t` non termina.

1.4 Interazione tra threads

1.4.1 BlockingQueue

E' una coda ThreadSafe, un produttore vi puo' inserire elementi finché essa non e' piena, il consumatore puo' rimuovere elementi dalla coda.

1.5 ThreadPools

Aprire un thread per ogni task e' poco efficiente. Thread frequenti e lightweight impattano negativamente sulle performance dell'applicazione.

Si hanno molti thread idle se il numero di essi supera il numero di processori disponibili.

In un ThreadPool si ha una coda di task, appena un thread si rende disponibile gi viene passato il task da eseguire. Il numero e il comportamento dei thread e' variabile e dipende dall'implementazione scelta.

1.5.1 CachedThreadPool

Se tutti i thread sono occupati e viene sottomesso un nuovo task si crea un nuovo thread. I thread restano attivi per 60 secondi dopo la terminazione del task, dopo i quali, se non viene sottomesso un nuovo task, vengono chiusi.

1.5.2 FixedThreadPool

Il numero di thread e' definito alla creazione. Se viene sottomesso una task e i thread sono tutti occupati, viene messo in una coda di dimensione illimitata.

1.5.3 ThreadPoolExecutor

Permette la creazione di un ThreadPool con un comportamento definibile alla creazione.

1.5.4 Terminazione di un executor

La terminazione puo' avvenire in modo

Graduale Tutti i task gia attivi finiscono l'esecuzione ma quelli nuovi vengono scartati. (`shutdown()`)

Istantaneo Tutti i thread vengono chiusi all'istante. (`shutdownNow()`)

1.6 Callable e Future

Gli oggetti di tipo `Runnable` non restituiscono valori di ritorno Un oggetto che implementa `Callable` invece definisce un task che puo' restituire un risultato e sollevare eccezioni. `Future` rappresenta il risultato di una `Callable` e definisce metodi per controllare la computazione.

Chapter 2

Condivisione delle risorse

Se piu' thread accedono alla stessa risorsa in modo concorrente e non ci sono controlli si verifica una race condition

2.1 Locks

Una Lock puo' essere bloccata al massimo da un thread; se un nuovo thread prova a richiedere la lock prima che venga sbloccata, viene fermata l'esecuzione del nuovo chiamante.

2.1.1 Deadlock

E' possibile che due thread si blocchino a vicenda.

2.2 Condition

Una condition permette di sospendere un thread (`await()`) e risvegliarlo (`notify()`) Vedere rimozioni concorrenti 3^a lezione

2.3 Sincronizzazione a alto livello

2.3.1 Blocchi sincronizzati

Quando si entra in un blocco

```
synchronized (object) {}
```

Il thread chiamante acquisisce una lock su `object` e la rilascia all'uscita dal blocco.

2.3.2 Metodi sincronizzati

```
public synchronized void doSomething() {}
```

Acquisisce una lock su `this`

2.4 Monitors

2.4.1 wait e notify

La chiamata di `wait()` sospende il thread in attesa di essere risvegliato tramite attesa passiva (niente polling). Il metodo `notify()` risveglia un thread precedentemente sospeso.

Per invocare questi metodi bisogna prima aver acquisito una lock sull'oggetto. Alla chiamata di `wait()` la lock viene rilasciata. Vedere esempio RWMonitor 4^a lezione.

2.5 Classi atomiche

Appositamente studiate per permettere l'accesso concorrente, non c'e' bisogno di utilizzare lock o metodi sincronizzati.

Chapter 3

IO

3.1 Stream

Uno stream e' una connessione tra un programma JAVA e un dispositivo esterno (File, connessione di rete...) Sono bloccanti e one-way.

3.2 Classe File

Fornisce, oltre alla path per l'individuazione del file, anche dei metodi per restituire meta-info sul file.

3.3 Buffered I/O Stream

Implementano una bufferizzazione per gli stream. I tempi di lettura e scrittura diminuiscono significativamente.

3.4 I/O StreamReader

Converte i byte non-unicode in caratteri unicode e viceversa.

3.5 Serializzazione / De-serializzazione

Un oggetto che implementa `Serializable` permette di essere inviato/salvato e riletto da un programma JAVA Il programma che intende de-serializzare un oggetto deve conoscere la classe. I campi marcati come `transient` non vengono serializzati.

3.5.1 Caching

Ogni volta che un oggetto viene serializzato e inviato ad una ObjectOutputStream, un suo riferimento viene memorizzato in una identity hash table.

Se l'oggetto viene scritto nuovamente viene inserito solamente un puntatore all'oggetto precedente.

3.5.2 Version Control

Per identificare la versione di un oggetto serializzato si utilizza il `serialVersionUID`. Se il SUID cambia, la de-serializzazione non e' possibile.

Chapter 4

NIO

4.1 Canali e Buffer

Un canale NIO e' analogo a uno stream IO. Tutti i dati inviati o letti da un canale devono essere memorizzati in un buffer.

Vedi variabili di stato: lezione del 22 Ottobre.

Un canale e' bidirezionale. I canali orientati alle connessione TCP e UDP possono essere impostati non bloccanti.

Chapter 5

JSON

E' un formato per l'interscambio di dati indipendente dalla piattaforma. E' basato su due strutture:

- Coppie (chiave - valore)
- Liste ordinate di valori

Le chiavi devono essere stringhe. I valori ammissibili sono

- String
- Number
- Object (JSON Object)
- Array
- Boolean
- null

Chapter 6

Le socket

6.1 La classe InetAddress

Rappresenta ad alto livello un indirizzo IP. Permette anche la risoluzione degli indirizzi attraverso il DNS tramite il metodo `getByName`

6.1.1 Caching

Gli indirizzi risolti tramite la InetAddress vengono salvati in una cache locale.

6.2 Tipi di socket

Esistono due tipi di socket server:

- welcome sockets
- connection sockets

Un client crea una active socket verso la welcome socket del server, il quale a sua volta creerà una connection socket con cui comunicare con in client.

6.3 Channel Multiplexing

Nel modello non-blocking la chiamata al sistema restituisce il controllo al chiamante prima che l'operazione sia *pianamente soddisfatta*