java.util.concurrent

Marco Danelutto
LPRb 2007-2008

Since 1.5 ...

- Package introdotto con la versione 1.5 di Java
- Contiene classi/package di supporto ad applicazioni concorrenti
- Due cose in particolare
  - BlockingQueue
  - ThreadPoolExecutor

BlockingQueue

- Interfaccia implementata da varie classi
  - attesa se prelievo su coda vuota
  - non può contenere valori null
  - può avere un limite sulla capacità (blocco sulla put, in caso)
  - implementazione thread safe (per le operazioni tipiche)

BlockingQueue

- LinkedBlockingQueue<E> queue = ...
- queue.put(E)
- E x = queue.take()
- altri metodi (non "tipici", vedi API doc)

Produttore consumatore

```
class Producer implements Runnable {
   private final BlockingQueue queue;
   Producer(BlockingQueue q) { queue = q; }
   public void run() {
     try {
       while(true) { queue.put(produce()); }
     } catch (InterruptedException ex) { ... handle ...}
   }
   Object produce() { ... }
 }
```

```
class Consumer implements Runnable {
  private final BlockingQueue queue;
  Consumer(BlockingQueue q) { queue = q; }
  public void run() {
    try {
      while(true) { consume(queue.take()); }
    } catch (InterruptedException ex) { ... handle ...}
  }
  void consume(Object x) { ... }
}
```

Executor (interface)

- Oggetto in grado di eseguire Runnable
  - Executor e ...
    e.execute(runnableTask1);
    ...
    e.execute(runnableTaskN);
- Implementation dependent
  - nello stesso thread del chiamante, in un

ExecutorService

- come un Executor
- metodo Future submit(Runnable) permette di sottomettere un task e di ottenere un'handle per controllarne lo stato
- Future<T>
  - f.cancel(), isCancelled(), isDone(), T f.get() (bloccante),

ThreadPool con Executors

- metodi statici della classe Executors
- Static ExecutorService Executors.newFixedThreadPool(int threads)
  - crea un thread pool con #thread dato
- Static ExecutorService Executors.newCachedThreadPool()
  - crea un thread pool: un nuovo thread per ogni nuovo tasks, se non posso riutilzzare

Caratteristiche

- fase di esecuzione
  - execute, submit
- fase di terminazione
  - shutdown(): niente più nuovi task da eseguire, termina thread appena possibile, completando task pendenti
  - shutdownNow(): niente più nuovi task, termi thread immediatamente (interrupt

Caratteristiche (2)

- isTerminated()
  - true se tutti i task sono terminati
- isShutdown()
  - true se la shutdown è terminata

ThreadPoolExecutor

- è un ExecutorService che esegue task Runnable utilizzando thread di un pool
- permette di definire parametri
  - #thread (core, max), tempo per cui vanno mantenuti attivi i thread senza task, coda per i task che non trovano un thread libero

ThreadPoolExecutor

- ThreadPoolExecutor(
  int corePoolSize, // #thread nel pool
  int maximumPoolSize, // max se task >>
  // quanto attendere prima di elim thread
  // inattivo fra core e max
  long keepAliveTime, TimeUnit unit,
  // coda per mantenere task senza thread
  BlockingQueue<Runnable> workQueue)

ThreadPoolExecutor pool size

- task (Runnable)
  - #thread < core => crezione
  - #thread > core & < max
    && coda piena => creazione

PoolSize (2)

- core = max
  - fixed size task pool
- max = Integer.MAX_VALUE
  - numero arbitrario di thread
- coda senza dim max (LinkedBlockingQueue ())

Creazione dei nuovi thread

- per default con
  - Executors.defaultThreadFactory()
  - crea thread non demoni con Thread.NORM_PRIORITY
- ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue, ThreadFactory threadFactory)

ThreadFactory

- interfaccia
- public Thread newThread(Runnable t);
- class SimpleThreadFactory implements ThreadFactory
  {
  public Thread newThread(Runnable r) {
  return new Thread(r);
  }
  }

tempi di keepAlive

- quando ho più thread di core (e < max)
- e thread inattivo
  - aspetto keepAlive timeUnit
  - poi termino il thread
  - se necessario, ne ricreo ...
  - e.g. (... , 5000, TimeUnit.MILLISECONDS, ...)

Gestione code

- si usa una BlockingQueue
- #thread < core
  - si crea un thread per un nuovo task
- #thread > core
  - si incoda un nuovo task
- #thread > core e < max e non ha successo l'incodamento
  - si crea un nuovo thread

Task non accettati

- task non accettati
  - e.g. coda piena e #thread = max
  - politiche predefinite. includono:
    - throw RejectedExecutionException
    - task eseguito da che fa la execute
    - scarta la richiesta
    - scarta la richiesta più vecchia e riprova

pre/post processing

- beforeExecute(java.lang.Thread, java.lang.Runnable)
  - metodo da sovrascrivere per controllare pre-esecuzione (quale thread esegue, cosa)
- afterExecute(java.lang.Runnable, java.lang.Throwable)
  - metodo da sovrascrivere per controllare post-esecuzione (cosa è terminato, come)

Quando si usano

- Executors.newFixedThreadPool()
  - quando si può adottare un Executor standard
- ThreadPoolExecutor
  - quando occore un controllo/tuning più fine sui/dei parametri