Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java: Thread Pool, Executor/5b71893317a1437e093bd567

TUTORIAL

Java: Thread Pool, Executor

Topics

The thread pool supporting classes are introduced in package java.lang.concurrent, in JDK 1.5.

Thread Pool

A thread pool is a managed collection of threads that are available to execute tasks. When a large number of tasks is executed using a thread pool, the performance improves as the threads are re-cycled to execute the tasks, which reduces the per-task invocation overhead.

To use a thread pool, you can use an implementation of the interface ExecutorService, such as ThreadPoolExecutor or ScheduledThreadPoolExecutor. However, more convenient factory methods are provided in the Executors class as follows:

- Executors.newSingleThreadExecutor(): creates a single background thread.
- Executors.newFixedThreadPool(int numThreads): creates a fixed size thread pool.
- Executors.newCachedThreadPool(): create a unbounded thread pool, with automatic thread reclamation.

The steps of using thread pool are:

1. Write you worker thread class which implements Runnable interface. The run() method specifies the behavior of the running thread.
2. Create a thread pool (ExecutorService) using one the factory methods provided by the Executors class. The thread pool could have a single thread, a fixed number of threads, or an unbounded number of threads.
3. Create instances of your worker thread class. Use execute(Runnable r) method of the thread pool to add a Runnable task into the thread pool. The task will be scheduled and executes if there is an available thread in the pool.

Executor interface

An Executor object can execute Runnable tasks submitted. The interface declares an abstract method:

```
public void execute(Runnable r)
```

It executes the given task at some time in the future. The task may be executed in a new thread, in a thread pool, or in the calling thread, depending on the implementation of `Executor` (e.g. single thread or thread pool)

ExecutorService interface

Interface ExecutorService declares many abstract methods. The important ones are:

```
public void shutdown();
   // Initiates an orderly shutdown of the thread pool.
   // The previously executed/submitted tasks are allowed to
complete,
   // but no new tasks will be scheduled.
public <T> Future<T> submit(Callable<T> task);
   // Submit or schedule the callable task for execution, which
returns a Future object.
```

Executors class

The class Executors provides factory methods for creating Executor object. For example:

```
static ExecutorService newSingleThreadExecutor()
static ExecutorService newFixedThreadPool(int nThreads)
static ExecutorService newCachedThreadPool()
static ScheduledExecutorService newSingleThreadScheduledExecutor()
static ScheduledExecutorService newScheduledThreadPool(int size)
```

For example,

Java

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class WorkerThread implements Runnable
{
   private int workerNumber;
   WorkerThread(int workerNumber)
   {
      this.workerNumber = workerNumber;
   }
   public void run()
   {      // The thread simply prints 1 to 5
      for (int i = 1; i <= 5; ++i)
      {
         System.out.printf("Worker %d: %d\n", workerNumber, i);
         try
         {
            Thread.sleep((int)(Math.random() * 50)); // sleep for 0 to 0.05 second
         }
         catch (InterruptedException e) {}
      }
   }
}

class Main
{
  public static void main(String[] args)
  {
```

```
    int numWorkers = 5;                 // Number of workers
    int threadPoolSize = 2;             // Thread pool size

    ExecutorService pool = Executors.newFixedThreadPool(threadPoolSize);
    WorkerThread[] workers = new WorkerThread[numWorkers];
    for (int i = 0; i < numWorkers; ++i)
    {
      workers[i] = new WorkerThread(i+1);
      pool.execute(workers[i]);
    }
    pool.shutdown();
  }
}
```

Worker 1 and 2 were first scheduled for execution using the 2 threads in the pool, followed by worker 3, 4 and 5. After the task using the thread completes, the thread is returned to the pool. Another task can then be scheduled and begin execution.

You can use **pool.shutdown()** to shutdown all the threads in the pool.