Java Thread Pool

Java Thread pool represents a group of worker threads that are waiting for the job and reuse many times.

In case of thread pool, a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

Advantage of Java Thread Pool

Better performance It saves time because there is no need to create new thread.

Real time usage

It is used in Servlet and JSP where container creates a thread pool to process the request.

Example of Java Thread Pool

Let's see a simple example of java thread pool using ExecutorService and Executors.

File: WorkerThread.java

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

class WorkerThread implements Runnable  {
   private String message;
   public WorkerThread(String s) {
      this.message=s;
   }

public void run() {
      System.out.println(Thread.currentThread().getName()+" (Start) message = "+mes
sage);
      processmessage();//call processmessage method that sleeps the thread for 2 sec

      System.out.println(Thread.currentThread().getName()+" (End)");//prints thread
                                                                    name
   }
```

```
    private void processmessage() {
        try {
                Thread.sleep(2000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
    }
}
```

File: JavaThreadPoolExample.java

```
public class TestThreadPool  {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(5);//creating a pool of
5 threads
        for (int i = 0; i < 10; i++) {
            Runnable worker = new WorkerThread("" + i);
            executor.execute(worker); //calling execute method of ExecutorService
          }

        executor.shutdown();
        while (!executor.isTerminated()) {   }

        System.out.println("Finished all threads");
    }
 }
```

Output:

```
pool-1-thread-1 (Start) message = 0
pool-1-thread-2 (Start) message = 1
pool-1-thread-3 (Start) message = 2
pool-1-thread-5 (Start) message = 4
pool-1-thread-4 (Start) message = 3
pool-1-thread-2 (End)
pool-1-thread-2 (Start) message = 5
pool-1-thread-1 (End)
pool-1-thread-1 (Start) message = 6
pool-1-thread-3 (End)
pool-1-thread-3 (Start) message = 7
pool-1-thread-4 (End)
```

```
pool-1-thread-4 (Start) message = 8
pool-1-thread-5 (End)
pool-1-thread-5 (Start) message = 9
pool-1-thread-2 (End)
pool-1-thread-1 (End)
pool-1-thread-4 (End)
pool-1-thread-3 (End)
pool-1-thread-5 (End)
Finished all threads
```