**Multithreading Enhancements – Part-04**

* **Thread Pools [Executor Framework]:**

Creating a new thread for every job may create performance and memory problems, to overcome this we should go for thread pool.

Thread Pool is a pool of already created threads ready to do our job.

Java 1.5 version introduced Thread Pool Framework to implement thread pools.

Thread Pool framework also known as Executor framework.

We can create a thread pool as follows

ExecutorService service = Executors.newFixedThreadPool(3);

We can submit a Runnable job by using submit() method

service.submit(job);

We can shutdown executor service by using shutdown() method

service.shutdown();

* **Example:**

import java.util.concurrent.\*;

class PrintJob implements Runnable{

String name;

PrintJob(String name){

this.name = name;

}

public void run(){

System.out.println(name+”… job started by Thread:”+Thread.currentThread().getName());

try{

} catch(InterruptedException ie){}

System.out.println(name+”…job completed by Thread:”+Thread.currentThread().getName());

}

}

class ExecutorDemo{

public static void main(String[] args){

PrintJob[] jobs = { new PrintJob(“durga”),

new PrintJob(“Ravi”),

new PrintJob(“shiva”)

new PrintJob(“pavan”)

new PrintJob(“suresh”)

new PrintJob(“anil”)};

ExecutorService service = Executors.newFixedThreadPool(3);

for(PrintJob printJob: jobs){

service.submit(job);

}

Service.shutdown();

}

}

Note:

I the above example three threads are responsible to execute six jobs. So that a single thread can be reused for multiple jobs.

While designing web servers and application servers, we can use thread pool concept.

* **Callable and Future:**

In the case of Runnable job thread won’t return anything after completing the job.

If a thread is required to return to some result after execution then we should go for callable.

Callable interface contains call

public Object call() throws Exception

If we submit a callable object to executor then after completing the job thread returns an object of the type Future. That is Future Object can be used to retrieve the result from callable job.

import java.util.concurrent.\*;

class MyCallable implements Callable{

int num;

MyCallable(int num){

this.num = num;

}

public Object call() throws Exception{

System.out.println(Thread.currentThread().getName()+”is … responsible to find sum of first”+num+”numbers”);

int sum =0;

for(int i = 1; 1 <= num; i++){

sum = sum + i;

}

return sum;

}

}

class CallableFutureDemo{

public static void main(String[] args) throws Exception{

MyCallable[] jobs = {new MyCallable(10),

new MyCallable(20),

new MyCallable(30),

new MyCallable(40),

new MyCallable(50),

new MyCallable(60)};

ExecutorService service = Executors.newFixedThreadPool(3);

for(MyCallable job: jobs){

Future f = service.submit(job);

System.out.println(f.get());

}

Service.shutdown();

}

}

Output:

55

210

465

820

1275

1830

* **Differences between Runnable and Callable Interface:**

|  |  |
| --- | --- |
| Runnable | Callable |
| If a thread is not required to return anything after completing the job then we should go for Runnable. | If a thread required to return something after completing the job, then we should go for callable. |
| Runnable interface contains only one method  public void run() | Callable interface contains only one method  public void call() |
| Runnalbe job not required to return anything and hence return type of run() is void. | Callable job is required to return something and hence return type of call() method is Object. |
| Within the run() if there is any chance of raising checked execption, compulsory we should handle by using try/catch because we can’t use throws keyword for run() method as it’s a overridden method. | Within call() method, if there is any chance of raising checked execption, we are not required to handle by try/catch because call() method already throws exception |
| Runnable interface present in java.lang package | Callable interface present java.util.concurrent package |
| Introduced in 1.0 version | Introduced in 1.5 version |