Java executor framework tutorial and best practices

MARCH 12, 2013 LOKESH 19 COMMENTS

Executors framework (java.util.concurrent.Executor), released with the JDK 5 in package java.util.concurrent is used to run the Runnable objects without creating new threads every time and mostly re-using the already created threads.

We all know about that there are two ways to create a thread in java. If you want to read more about their comparison, **read this post**. Creating a thread in java is a very expensive process which includes memory overhead also. So, it's a good idea if we can re-use these threads once created, to run our future runnables.

In this post, I will write some demo programs demonstrating the usage of Executor and then we will talk about some best practices which you need to keep in mind while designing your next multi-threaded application.

If you want read more about other **multi-threading** topics, **follow this link**.

Basic usage demo application

In our demo application, we have two tasks running. Neither is expected to terminate, and both should run for the duration of the application's life. I will try to write a main wrapper class such that:

If any task throws an exception, the application will catch it and restart the task.
If any task runs to completion, the application will notice and restart the task.

Below is the code sample of above required application:

```
1  package com.howtodoinjava.multiThreading.executors;
2
3  import java.util.concurrent.ExecutorService;
4  import java.util.concurrent.Executors;
5  import java.util.concurrent.Future;
6
7  public class DemoExecutorUsage {
8
9      private static ExecutorService executor = null;
10     private static volatile Future taskOneResults = null;
11     private static volatile Future taskTwoResults = null;
12
13     public static void main(String[] args) {
14         executor = Executors.newFixedThreadPool(2);
15         while (true)
16         {
17             try
18             {
19                 checkTasks();
20                 Thread.sleep(1000);
21             } catch (Exception e) {
22                 System.err.println("Caught exception: " + e.getMessage());
23             }
24         }
25     }
```

```
26
27      private static void checkTasks() throws Exception {
28          if (taskOneResults == null
29                  || taskOneResults.isDone()
30                  || taskOneResults.isCancelled())
31          {
32              taskOneResults = executor.submit(new TestOne());
33          }
34
35          if (taskTwoResults == null
36                  || taskTwoResults.isDone()
37                  || taskTwoResults.isCancelled())
38          {
39              taskTwoResults = executor.submit(new TestTwo());
40          }
41      }
42  }
43
44  class TestOne implements Runnable {
45      public void run() {
46          while (true)
47          {
48              System.out.println("Executing task one");
49              try
50              {
51                  Thread.sleep(1000);
52              } catch (Throwable e) {
53                  e.printStackTrace();
54              }
55          }
56
57      }
58  }
59
60  class TestTwo implements Runnable {
61      public void run() {
62          while (true)
63          {
64              System.out.println("Executing task two");
65              try
66              {
```

How to use BlockingQueue and ThreadPoolExecutor in java

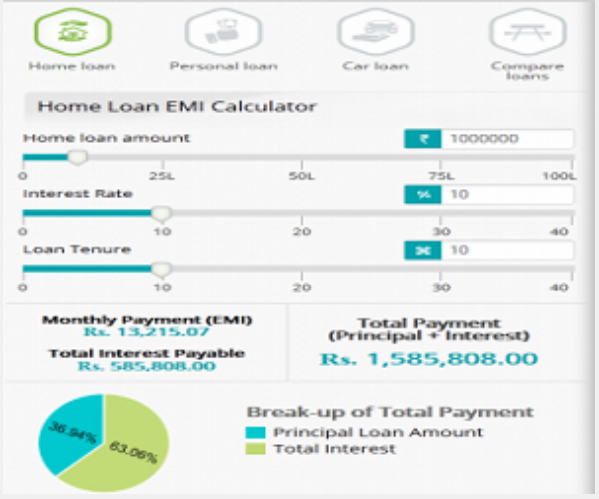
Difference between “implements Runnable” and “extends Thread” in java

When to use CountDownLatch : Java concurrency example tutorial

Throttling task submission rate using ThreadPoolExecutor and Semaphore

Fork/Join Framework Tutorial: ForkJoinPool Example

Writing a deadlock and resolving in java



```
                Thread.sleep(1000);
            } catch (Throwable e) {
                e.printStackTrace();
            }
        }
    }
}
```

Please do not forget to read best practices at the end of post.

Executing multiple tasks in a single thread

It's not necessary that each Runnable should be executed in a separate thread. Sometimes, we need to do multiple jobs in a single thread and each job is instance of Runnable. To design this type of solution, a multi runnable should be used. This multi runnable is nothing but a collection of runnables which needs to be executed. Only addition is that this multi runnable is also a Runnable itself.

Below is the list of tasks which needs to be executed in a single thread.

```
package com.howtodoinjava.multiThreading.executors;

public class TaskOne implements Runnable {
    @Override
    public void run() {
        System.out.println("Executing Task One");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class TaskTwo implements Runnable {
    @Override
    public void run() {
        System.out.println("Executing Task Two");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

```
}

public class TaskThree implements Runnable {
    @Override
    public void run() {
        System.out.println("Executing Task Three");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Lets create out multi runnable wrapper of above Tasks.

```
package com.howtodoinjava.demo.multiThread;

import java.util.List;

public class MultiRunnable implements Runnable {

    private final List<Runnable> runnables;

    public MultiRunnable(List<Runnable> runnables) {
        this.runnables = runnables;
    }

    @Override
    public void run() {
        for (Runnable runnable : runnables) {
             new Thread(runnable).start();
        }
    }
}
```

Now above multi runnable can be executed this way as in below program:

```
package com.howtodoinjava.demo.multiThread;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.RejectedExecutionHandler;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

```

```
public class MultiTaskExecutor {

    public static void main(String[] args) {

        BlockingQueue<Runnable> worksQueue = new ArrayBlockingQueue<Runnable>(
        RejectedExecutionHandler rejectionHandler = new RejectedExecutionHande
        ThreadPoolExecutor executor = new ThreadPoolExecutor(3, 3, 10, TimeUni

        executor.prestartAllCoreThreads();

        List<Runnable> taskGroup = new ArrayList<Runnable>();
        taskGroup.add(new TestOne());
        taskGroup.add(new TestTwo());
        taskGroup.add(new TestThree());

        worksQueue.add(new MultiRunnable(taskGroup));
    }
}

class RejectedExecutionHandelerImpl implements RejectedExecutionHandler
{
    @Override
    public void rejectedExecution(Runnable runnable,
            ThreadPoolExecutor executor)
    {
        System.out.println(runnable.toString() + " : I've been rejected ! ");
    }
}
```

Best practices which must be followed

1. Always run your java code against static analysis tools like PMD and FindBugs to look for deeper issues. They are very helpful in determining ugly situations which may arise in future.
2. Always cross check and better plan a code review with senior guys to detect and possible deadlock or livelock in code during execution. Adding a health monitor in your application to check the status of running tasks is an excellent choice in most of the scenarios.
3. In multi-threaded programs, make a habit of catching errors too, not just exceptions. Sometimes unexpected things happen and Java throws an error at you, apart from an exception.
4. Use a back-off switch, so if something goes wrong and is non-recoverable, you don't escalate the situation by eagerly starting another loop. Instead, you need to wait until the situation goes back to normal and then start again.
5. Please note that the whole point of executors is to abstract away the specifics of execution, so ordering is not guaranteed unless explicitly stated.

Happy Learning !!

Related Posts:

1. How to use BlockingQueue and ThreadPoolExecutor in java
2. Difference between “implements Runnable” and “extends Thread” in java
3. When to use CountDownLatch : Java concurrency example tutorial
4. Throttling task submission rate using ThreadPoolExecutor and Semaphore
5. Fork/Join Framework Tutorial: ForkJoinPool Example
6. Writing a deadlock and resolving in java

BEST PRACTICES EXECUTOR MULTI THREADING THREAD IN JAVA

PREVIOUS POST

Difference between “implements Runnable” and “extends Thread” in java

NEXT POST

Suppressed exceptions in java 7

19 THOUGHTS ON “JAVA EXECUTOR FRAMEWORK TUTORIAL AND BEST PRACTICES”

Gaurav

SEPTEMBER 25, 2014 AT 11:23 AM

Hi,

In your example DemoExecutorUsage, why you have created Future class objects, where no class implements Callable interface?

REPLY

★ Lokesh

SEPTEMBER 25, 2014 AT 12:13 PM

Hi Gaurav, Thanks for asking this very good question. In my understanding, Callable is different from Runnable in only one sense i.e. it can return some value. Otherwise they may be used alternatively in any similar situation without breaking the system. Please refer to **ExecutorService.submit(Runnable)** method. I am using same method it just for checking the status of running tasks, without expecting any rerun value from them.

I modified first example a bit and now observe the outputs.

```
1  public class DemoTaskExecutorVersion2 {
2
3      private static ExecutorService executor = null;
4      private static volatile Future taskOneResults = null;
5      private static volatile Future taskTwoResults = null;
6
7      public static void main(String[] args) {
8          executor = Executors.newFixedThreadPool(2);
9          while (true)
10         {
11             try
12             {
13                 checkTasks();
14                 Thread.sleep(1000);
```

```
            } catch (Exception e) {
                System.err.println("Caught exception: " + e.get
            }
        }
    }

    private static void checkTasks() throws Exception {
        if (taskOneResults == null
                || taskOneResults.isDone()
                || taskOneResults.isCancelled())
        {
            System.out.println("Starting task one.......");
            taskOneResults = executor.submit(new TestOne());
        }

        if (taskTwoResults == null
                || taskTwoResults.isDone()
                || taskTwoResults.isCancelled())
        {
           System.out.println("Starting task two.......");
            taskTwoResults = executor.submit(new TestTwo());
        }
    }
}

class TestOne implements Runnable {
    @Override
   public void run() {
        {
            System.out.println("Executing task one");
            try
            {
                Thread.sleep(5000);
            } catch (Throwable e) {
                e.printStackTrace();
            }
            System.out.println("Task one completed");
        }

    }
}

class TestTwo implements Runnable {
    @Override
   public void run() {
        {
            System.out.println("Executing task two");
            try
            {
                Thread.sleep(8000);
            } catch (Throwable e) {
```

```
                e.printStackTrace();
            }
            System.out.println("Task two completed");
        }
    }
}

Output:

Starting task one.......
Starting task two.......
Executing task one
Executing task two

Task one completed
Starting task one.......

Executing task one

Task two completed
Starting task two.......

Executing task two

Task one completed
Starting task one.......

Executing task one
```

As soon as any task is completed, application is able to identify it and restart it; that's whole purpose of this exercise. Monitor threads should never stop, and if they are stopped somehow, then application should start them again.

I hope that I am making sense.

REPLY

Hitmannura Salk

JULY 24, 2014 AT 10:37 AM

i did it but not working, so i post the question here, i code in eclipse-kepler for core java

REPLY

Hitmannura Salk

JULY 24, 2014 AT 9:52 AM

got output :
Executing Task Two
Executing Task One
Executing Task Three
was successful ,
but not exiting programme after Executing all the Threads needed.

REPLY

★ Lokesh

JULY 24, 2014 AT 10:05 AM

Use System.exit(0).

REPLY

Ram Krishna Dixit

JUNE 26, 2014 AT 4:27 PM

Hi Lokesh,
Very Informative topic however when I am executing given example code (Class DemoExecutorUsage) it continuously run and throws NullPointerException . There should be not null checks.

REPLY

★ Lokesh

JUNE 26, 2014 AT 5:52 PM

Yes it continuously runs.. but NullPointerException should not happen. Where it occur? Can you please share some exception trace.

REPLY

Rampal

JUNE 17, 2014 AT 9:05 AM

Hi Lokesh,
What would be the difference between these two code snippets :
worksQueue.add(new MultiRunnable(taskGroup));
and
executor.execute(new MultiRunnable(taskGroup));

Both of these would give the similar end results

REPLY

★ Lokesh

JUNE 17, 2014 AT 9:15 AM

Yes, they would give the similar end results.

REPLY

shoumesh

FEBRUARY 9, 2014 AT 9:30 AM

Lokesh Please Explain the Executor framework part of this program

REPLY

★ Lokesh

FEBRUARY 9, 2014 AT 4:02 PM

I will come up in a different post soon.

REPLY

Manish

OCTOBER 14, 2013 AT 11:05 AM

Hey Lokesh,
I have few doubts, please help me understand.
(1) we have created one object of blockingQueue and used that in

ThreadPoolExecutor, so what role it is playing in our current example.
(2) What I think is we should not use MultiRunnable class and instead of below calls
taskGroup.add(new TaskOne());
taskGroup.add(new TaskTwo());
taskGroup.add(new TaskThree());
executor.execute(new MultiRunnable(taskGroup));
we should use
executor.execute(new TaskOne());
executor.execute(new TaskTwo());
executor.execute(new TaskThree());

Because “runnable.run() in MultiRunnable” will execute the tasks within current parent thread instead of spawning new threads. Creating new threads in MultiRunnable will altogether means not utilizing the power of our Executor framework.

Please let us know your expert thoughts.

Regards,
Manish Kr Agarwal
manish.java@outlook.com

REPLY

★ **Lokesh Gupta**

OCTOBER 14, 2013 AT 10:57 PM

Hey Manish, thanks for your comment here.

I just looked again at code and realized it should been updated when I replied to mateen27. I don’t know how i missed here, because I usually make code changes immediately if I like the suggestion.

I believe, both of your concerns should be resolved with latest code.

REPLY

Pingback: Java executor framework tutorial and best pract...

mauro

SEPTEMBER 16, 2013 AT 2:53 PM

Please read at
http://www.javaswingcomponents.com/blog/taming-swing-threads-part-3-swing-worker

I would understand the use of swingworker .

Into the link above it says :

——————
Java 6 Release 18 and the SwingWorker.I have been working with the SwingWorker successfully since it was originally released in java 5. However there is a defect affecting the SwingWorker in Java 6 Release 18, which effectively brought a critical production system to its knees. The problem manifested in Java restricted the number of SwingWorkers that could be created. After launching a number of SwingWorkers, their thread pool reached capacity and no more workers could be created. The application became unresponsive as it needed SwingWorkers to fetch its required, however no new Workers could be created. Even though this problem was extremely serious, the workaround is very simplistic. Instead of calling the execute() method on the SwingWorker, simply pass the SwingWorker to your own Executor. executor.execute(swingWorker);

———————

Whi?

I have toi use always the Executor class for resolve the problem ?

You can please explain me how instantitate the executor into that case?

Tank you .
Mauro

REPLY

mateen27

MARCH 16, 2013 AT 10:43 PM

here is the link

https://docs.google.com/file/d/0B0_wI0TkntqKeTMxeWgtWHFnakE/edit?pli=1
it has src dir only with all java files…

Please let me know if its working (because i have written an swing application application using threads which is crashing so i want to use this concept… as runnable will resrart even if it crashes)

Regards.
Mateen SA.

REPLY

Lokesh Gupta

MARCH 17, 2013 AT 7:39 AM

I tried a few code changes and changing new Thread(runnable).start(); in place of runnable.run(); do the magic you want.

```
public class MultiRunnable implements Runnable {

private final List runnables;

public MultiRunnable(List runnables) {
this.runnables = runnables;
}

@Override
public void run() {
for (Runnable runnable : runnables) {
```

```
new Thread(runnable).start();
//runnable.run();
}
}
}
```

REPLY

mateen27

MARCH 14, 2013 AT 11:15 PM

Hi Lokesh,

if i am not wrong…
the multi executor is runnable and its executing other runnables

the following is the output i am getting…. when i execute multi executor (second) program

Executing task one
Executing task one
Executing task one …

however i need this program bt its not working fine… means executing only thread one… not thread two or three…

is thr any way to make this work…. ?

Regards,
Mateen SA

REPLY

Lokesh Gupta

MARCH 14, 2013 AT 11:32 PM

Hi, Yes multi runnable is group of runnables.

Can you please post the complete code of all your classes so that i can better be able to help you out??

REPLY

Note:- In comment box, please put your code inside **[java] ... [/java]** OR **[xml] ... [/xml]** tags otherwise it may not appear as intended.

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Are you Human and NOT robot? Solve this to prove !! *

seven − 2 =

Comment

You may use these HTML tags and attributes:

```
<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>
```

POST COMMENT

☐ Notify me of follow-up comments by email.

▣ Notify me of new posts by email.

SUBSCRIBE TO BLOG VIA EMAIL

Join 3,667 other subscribers

Email Address

SUBSCRIBE

REFERENCES

Oracle Java Docs

Spring 3 Reference

Spring 4 References

Jboss RESTEasy JAX-RS

Hibernate Developer Guide

Junit Wiki

Maven FAQs

Dzone Dev Links

JSP Homepage

ANT Homepage

META LINKS

Advertise

Share your knowledge

Privacy policy

Contact Us

About Us

POPULAR POSTS

Spring 3 and hibernate integration tutorial with example

Reading/writing excel files in java : POI tutorial

How hashmap works in java

Singleton design pattern in java

Useful java collection interview questions

Understanding hibernate first level cache with example

How hibernate second level cache works?

Working with hashCode and equals methods in java

How to make a java class immutable

4 ways to schedule tasks in Spring 3 : @Scheduled example