How to use BlockingQueue and ThreadPoolExecutor in java

OCTOBER 20, 2012 LOKESH 30 COMMENTS

Life has become very easy for java programmers working on multi-threaded applications after release of JDK 5. JDK 5 brought many features related to multi-threaded processing which were kind of nightmare for application developers, and even worse for those who had to debug this code in future for bug fixing. Sometimes, this resulted in deadlock situations as well.

In this post, i will suggest to use such a new feature ThreadPoolExecutor in combination with

In this post, I will suggest to use such a new feature ThreadPoolExecutor in combination with BlockingQueue. I will let you know the best practices to use above classes in your application.

Sections in this post:

- Introducing DemoTask
- Adding CustomThreadPoolExecutor
- Explaining BlockingQueue
- Explaining RejectedExecutionHandler
- Testing our code

Introducing DemoTask

I will not take much time here as our DemoTask is just another test thread written to support our logic and code.

```
1  package corejava.thread;
2
3  public class DemoThread implements Runnable
4  {
5      private String name = null;
6
7      public DemoThread(String name) {
8          this.name = name;
9      }
10
11     public String getName() {
12         return this.name;
13     }
14
15     @Override
16     public void run() {
17         try {
18             Thread.sleep(500);
19         } catch (InterruptedException e) {
20             e.printStackTrace();
21         }
22         System.out.println("Executing : " + name);
23     }
24 }
```


YOU MAY ALSO LIKE

Throttling task submission rate using ThreadPoolExecutor and Semaphore

Adding CustomThreadPoolExecutor

This is important. Our CustomThreadPoolExecutor is extension of ThreadPoolExecutor. Even without extending the ThreadPoolExecutor, simply creating its instance and using it, will also work correctly. But, we will miss some extremely useful features in terms of control of execution.

ThreadPoolExecutor provides two excellent methods which i will highly recommend to override i.e. beforeExecute() and afterExecute() methods. They provide very good handle on execution life cycle of runnables to be executed. Lets see above methods inside our CustomThreadPoolExecutor.

```
1  package corejava.thread;
2
3  import java.util.concurrent.BlockingQueue;
4  import java.util.concurrent.ThreadPoolExecutor;
5  import java.util.concurrent.TimeUnit;
6
7  public class CustomThreadPoolExecutor extends ThreadPoolExecutor {
8
9      public CustomThreadPoolExecutor(int corePoolSize, int maximumPoolSize,
10             long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQue
11         super(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue);
12     }
13
14     @Override
15     protected void beforeExecute(Thread t, Runnable r) {
16         super.beforeExecute(t, r);
17         System.out.println("Perform beforeExecute() logic");
18     }
19
20     @Override
```

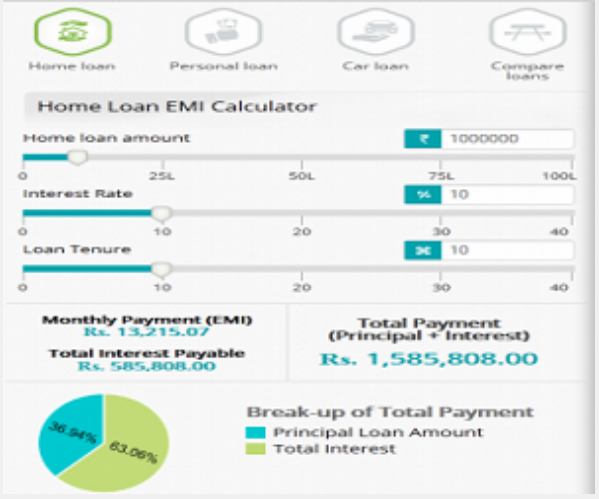

Java executor framework tutorial and best practices

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

When to use CountDownLatch : Java concurrency example tutorial

Inter-thread communication using piped streams in java

Writing a deadlock and resolving in java



```
21      protected void afterExecute(Runnable r, Throwable t) {
22          super.afterExecute(r, t);
23          if (t != null) {
24              System.out.println("Perform exception handler logic");
25          }
26          System.out.println("Perform afterExecute() logic");
27      }
28
29  }
```

Explaining BlockingQueue

If you remember solving the producer-consumer problem, before JDK 5, consumer had to wait until producer put something in resource queue. This problem can be easily solved using new BlockingQueue.

BlockingQueue is like another Queue implementations with additional capabilities. Any attempt, to retrieve something out of it, can be seen safe as it will not return empty handed. Consumer thread will automatically wait until BlockingQueue is not populated with some data. Once it fills, thread will consume the resource.

BlockingQueue works on following rules:

- If fewer than corePoolSize threads are running, the Executor always prefers adding a new thread rather than queuing.
- If corePoolSize or more threads are running, the Executor always prefers queuing a request rather than adding a new thread.
- If a request cannot be queued, a new thread is created unless this would exceed maximumPoolSize, in which case, the **task will be rejected**.

Explaining RejectedExecutionHandler

So the danger is, a task can be rejected as well. We need to have something in place to resolve this situation because no one would like to miss any single job in his application.

Can we do something about it? Yes, we can...[Borrowed from Obama]

BlockingQueue in case of rejection throws RejectedExectionException, we can add a handler for it.

Adding RejectedExecutionHandler is considered a good practice when using new concurrent APIs.

Testing our code

I am done with talking and its time to see if actually, what i said, works? Lets write a test case.

We have some 100 tasks. We want to run them using ideally 10, and maximum 20 threads. I am trying to write code as below. You might write it better or you have this solution.

```
1  package corejava.thread;
2
3  import java.util.concurrent.ArrayBlockingQueue;
4  import java.util.concurrent.BlockingQueue;
5  import java.util.concurrent.RejectedExecutionHandler;
6  import java.util.concurrent.ThreadPoolExecutor;
7  import java.util.concurrent.TimeUnit;
8
9  public class DemoExecutor
10 {
11     public static void main(String[] args)
12     {
13         Integer threadCounter = 0;
14         BlockingQueue<Runnable> blockingQueue = new ArrayBlockingQueue<Runnabl
15
16         CustomThreadPoolExecutor executor = new CustomThreadPoolExecutor(10,
17                                             20, 5000, TimeUnit.MILLISECONDS, k
18
19         executor.setRejectedExecutionHandler(new RejectedExecutionHandler() {
20             @Override
21             public void rejectedExecution(Runnable r,
22                     ThreadPoolExecutor executor) {
23                 System.out.println("DemoTask Rejected : "
24                         + ((DemoThread) r).getName());
25                 System.out.println("Waiting for a second !!");
26                 try {
```

```
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Lets add another time : "
                        + ((DemoThread) r).getName());
                executor.execute(r);
            }
        });
        // Let start all core threads initially
        executor.prestartAllCoreThreads();
        while (true) {
            threadCounter++;
            // Adding threads one by one
            System.out.println("Adding DemoTask : " + threadCounter);
            executor.execute(new DemoThread(threadCounter.toString()));

            if (threadCounter == 100)
                break;
        }
    }

}
```

Execute above code and you will see the result is as desired and performance is also good.

I hope i am able to make a point here. Please let me know of your thoughts also.

Happy Learning !!

Related Posts:

1. Throttling task submission rate using ThreadPoolExecutor and Semaphore
2. Java executor framework tutorial and best practices
3. Difference between “implements Runnable” and “extends Thread” in java
4. When to use CountDownLatch : Java concurrency example tutorial
5. Inter-thread communication using piped streams in java
6. Writing a deadlock and resolving in java

BLOCKING QUEUE EXECUTOR INTER THREAD COMMUNICATION MULTI THREADING REJECTION HANDLER THREAD POOL

PREVIOUS POST

Auto reload configuration when any change happen : part 2

NEXT POST

When to use comparable and comparator interfaces in java

30 THOUGHTS ON “HOW TO USE BLOCKINGQUEUE AND THREADPOOLEXECUTOR

IN JAVA”

Vishnu

APRIL 10, 2014 AT 7:50 AM

This line giving error :
BlockingQueue blockingQueue = new ArrayBlockingQueue(
50);

REPLY

Ken

SEPTEMBER 12, 2014 AT 8:00 AM

Yes, this line gives an error.

REPLY

★ Lokesh

SEPTEMBER 12, 2014 AT 8:24 AM

I am using this constructor.

REPLY

RP

SEPTEMBER 16, 2014 AT 7:13 AM

Can i use LinkedBlockingQueue like LinkedBlockingQueue logQueue = new LinkedBlockingQueue(QUEUE_CAPACITY)?

REPLY

★ Lokesh

SEPTEMBER 16, 2014 AT 7:39 AM

I do not see any problem as far as it is implementation of BlockingQueue.

REPLY

TechSawi

JANUARY 23, 2014 AT 1:47 PM

How to maintain the thread execution order? I want threads to be executed in the order they are created.

REPLY

★ Lokesh

JANUARY 23, 2014 AT 2:48 PM

Two thoughts come up immediately in my mind.
1) If you want ordering, then no need to multi-thread your application. Multi-threading is inherently un-ordered and for parallel execution.
2) If you still demand ordering based on object counter, doesn't Queue process them sequentially in order they were added? Yes it does.

Am I missing anything, Guys?

REPLY

Rampal

JUNE 17, 2014 AT 6:23 AM

I think they are not executing in the same order as they are created. For example, in my first run of the above code, thread 5 got executed before thread 1

REPLY

Guru

JANUARY 9, 2014 AT 4:41 PM

Hi Lokesh, Thanks for the beautiful explanation!!

blockingQueue has never been used, why have you declared in first place?

REPLY

★ **Lokesh Gupta**

JANUARY 11, 2014 AT 11:54 AM

I passed it into CustomThreadPoolExecutor's constructor.

REPLY

Matt Hoover

DECEMBER 18, 2013 AT 8:20 AM

Your main program never terminates.

REPLY

★ **Lokesh Gupta**

DECEMBER 18, 2013 AT 10:43 AM

Yes, it's expected and intended behavior.

REPLY

Dimal

DECEMBER 17, 2013 AT 2:56 PM

I have a scenario where I need a module based scheduling.. Probably I will consider separate BLOCKING QUEUE for each module for handling different their implementation. Do you think that creating a Map for module name to its Blocking queue would be the right solution ?

REPLY

★ **Lokesh Gupta**

DECEMBER 17, 2013 AT 10:18 PM

Seems ok to me. Basically you would like to hide it behind a factory kind of wrapper. It will help in replacing the logic if not proved correct in future.

REPLY

dimal

FEBRUARY 4, 2014 AT 12:25 PM

Is there any way that I can override getTask() method of threadpool executor ?. Because I need to implement my own logic to get next task.

Thanks,
Dimal

REPLY

★ Lokesh

FEBRUARY 4, 2014 AT 1:00 PM

NO. You can not. Reason is simple in two aspects. First the method getTask() is private so you cannot override. Second, this method plays a critical part in fetching the next executable/runnable from queue. If you are changing the logic here, then the whole idea of using a Queue falls apart. Queues use a fixed insertion/retrieval ordering and any program should honor this default behavior.

Regarding your own logic to get next task, I will suggest you to research more on PriorityBlockingQueue concept. You can add a comparator to priority queue and it will re-arrange (theoretically i suppose, not tested myself) the runnables in queue and whenever executor ask for new task, queue can provide the next task as per your logic in comparator.

Again, I have not tested it myself but this should be your area of research.

http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/PriorityBlockingQueue.html

REPLY

dimal

FEBRUARY 4, 2014 AT 1:41 PM

Thanks Lokesh for your valuable feedback. I have a question about the second point that you mentioned, why do we need a complex method to fetch next executable from queue? . So you are saying if I have my own implementation for getTask(), I no need to extends ThreadPoolExecutor.

Thanks,
Dimal

REPLY

Bala

NOVEMBER 22, 2013 AT 12:24 PM

Hi,
Is there a possibility that I can reduce the number of threads of executor service once the threads are started? executor.shutdown() will terminate all the threads, but can I terminate one of the thread which was started by executor service?

Thanks,
Bala

REPLY

AK

OCTOBER 20, 2013 AT 9:52 PM

Hi,

I am facing a problem in ThreadPoolExecutor, i am using ArrayBlockingQueue to queue the tasks. CorePoolSize is 50, maximum pool size is MAX value of integer. When I submit requests and number of threads reaches corePoolSize then it queue up the request in the ArrayBlockingQueue but never execute the run() method of the submitted requests. Could you please let me know what could be reason for this and possible solution. I'm facing this problem due to which my application is getting timeout as it does not get the response.

Thanks
AK

REPLY

★ Lokesh Gupta

OCTOBER 20, 2013 AT 10:56 PM

Because your tasks are not getting executed. Try to put some log statements OR sysout statements to check that even a single task was executed.
Then check if they are not being rejected. Add RejectedExecutionHandler.

If still struck, paste the code here. I will try to get you a solution.

REPLY

Desmond

OCTOBER 11, 2013 AT 4:12 AM

I still don't see why new tasks should be rejected if the queue is full. If new tasks are rejected when the queue is full, the purpose of using bounded queue is lost. The purpose of the bounded queue is that the put() method is blocked when the queue is full. It seems the ThreadPoolExecutor is not using this feature.

Therefore, I think this is a design fault in ThreadPoolExecutor.

REPLY

★ Lokesh Gupta

OCTOBER 11, 2013 AT 9:59 AM

Hmmm, it's good point. But I do not completely agree with you. Reason is that this is well documented behavior. Please refer to bounded queue and unbounded queue sections.

http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/concurrent/ThreadPoolExecutor.java

It is desired in many cases to prevent resource exhaustion. Otherwise waiting queue can keep growing and eat up whole system resources.

REPLY

Manish

OCTOBER 13, 2013 AT 5:10 PM

New tasks submitted in method execute(java.lang.Runnable) will be rejected when the Executor has been shut down, and also when the Executor uses finite bounds for both maximum threads and work queue capacity, and is saturated

REPLY

★ Lokesh Gupta

OCTOBER 13, 2013 AT 9:15 PM

100% correct. You are right.

REPLY

Matt Hoover

DECEMBER 18, 2013 AT 8:09 AM

Try using the CallerRunsPolicy:
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.CallerRunsPolicy.html

From ThreadPoolExecutor JavaDoc:
“In ThreadPoolExecutor.CallerRunsPolicy, the thread that invokes execute itself runs the task. This provides a simple feedback control mechanism that will slow down the rate that new tasks are submitted.”
It also ensures that tasks will never be discarded.

REPLY

Lokesh

MAY 23, 2014 AT 8:10 AM

It’s good point.

REPLY

Vishesh

JULY 5, 2013 AT 12:20 PM

Using RejectedExecutionHandler is good concept. Liked it

REPLY

shai

JUNE 29, 2014 AT 2:41 PM

The RejectedExecutionHandler has one approach problem. When a very large number of executions come in at a very short time (lets say the implementation is a kind of server) the method may encounter a stack overflow as rejected calls execution which rejects the execution back to rejected and so on eventually reaching overflow. I write this from experience using such an implementation in high volume transaction system. Thinking of a different solution as I can’t loose executions.

REPLY

★ Lokesh

JUNE 29, 2014 AT 6:28 PM

Pretty valid point. Any suggestion what you would like to do in this case? One solution may be increasing the number of size of thread pool or queue size, but it has it's own disadvantages in resource critical systems. Another approach may be to let tasks reject and log them somewhere to get a report in future.

REPLY

Pingback: Inter-thread communication using piped streams in java « How to do in "JAVA"

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 × = 56

Comment

You may use these HTML tags and attributes:

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

POST COMMENT

☐ Notify me of follow-up comments by email.

▣ Notify me of new posts by email.

SUBSCRIBE TO BLOG VIA EMAIL

Join 3,665 other subscribers

Email Address

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