Throttling task submission rate using ThreadPoolExecutor and Semaphore

MAY 24, 2014 LOKESH 4 COMMENTS

If you may know that in web-servers you can configure the maximum number of concurrent connections to the server. If more connections than this limit come to server, they have to wait until some other connections are freed or closed. This limitation can be taken as throttling. Throttling is the capability of regulating the rate of input for a system where output rate is slower than input. It is necessary to stop the system from crashing or resource exhaustion.

In one of my previous post related to `BlockingQueue` and `ThreadPoolExecutor`, We learned about creating a `CustomThreadPoolExecutor` which had following capabilities:

1) Tasks being submitted to blocking queue
2) An executor which picks up the task from queue and execute them
3) Had overridden **beforeExecute()** and **afterExecute()** methods to perform some extra activities if needed
4) Attached a `RejectedExecutionHandler` which handle the task if it got rejected because the queue was full

Our approach was good enough already and capable of handling most of the practical scenarios. Now let's add one more concept into it which may prove beneficial in some conditions. This concept is around throttling of task submission in queue.

In this example, throttling will help in keeping the number of tasks in queue in limit so that no task get rejected. It essentially removes the necessity of `RejectedExecutionHandler` as well.

Previous Solution Using CustomThreadPoolExecutor with RejectedExecutionHandler

In this solution, we had following classes:

DemoTask.java

```
1  public class DemoTask implements Runnable
2  {
3     private String name = null;
4
5     public DemoTask(String name) {
6        this.name = name;
7     }
8
```

```
    public String getName() {
        return this.name;
    }

    @Override
    public void run(){
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("Executing : " + name);
    }
}
```

CustomThreadPoolExecutor.java

```
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

public class CustomThreadPoolExecutor extends ThreadPoolExecutor
{
    public CustomThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long
                                    TimeUnit unit, BlockingQueue<Runnable> wor
    {
        super(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue);
    }

    @Override
    protected void beforeExecute(Thread t, Runnable r)
    {
        super.beforeExecute(t, r);
    }

    @Override
    protected void afterExecute(Runnable r, Throwable t)
    {
        super.afterExecute(r, t);
        if (t != null)
        {
            t.printStackTrace();
        }
    }
}
```

DemoExecutor.java

How to use BlockingQueue and ThreadPoolExecutor in java

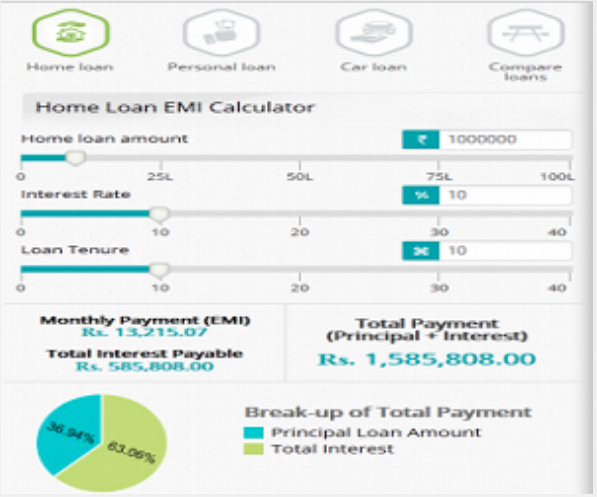
Java executor framework tutorial and best practices

When to use CountDownLatch : Java concurrency example tutorial

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

2 ways to execute timer tasks in spring 3

Inter-thread communication using piped streams in java



```
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 DemoExecutor
{
   public static void main(String[] args)
   {
      Integer threadCounter = 0;
      BlockingQueue<Runnable> blockingQueue = new ArrayBlockingQueue<Runnable>
      CustomThreadPoolExecutor executor = new CustomThreadPoolExecutor(10, 20,
      executor.setRejectedExecutionHandler(new RejectedExecutionHandler()
         {
            @Override
            public void rejectedExecution(Runnable r, ThreadPoolExecutor execu
            {
               System.out.println("DemoTask Rejected : " + ((DemoTask) r).getN
               try
               {
                  Thread.sleep(1000);
               } catch (InterruptedException e)
               {
                  e.printStackTrace();
               }
               System.out.println("Lets add another time : " + ((DemoTask) r).
               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 DemoTask(threadCounter.toString()));
         if (threadCounter == 1000)
            break;
      }
   }
}
```

If we run the above program, we will get the **output** like below:

```
DemoTask Rejected : 71
Executing : 3
Executing : 5
...
```

```
...
```

There will be multiple occurrences of "***DemoTask Rejected***". In next solution, we will put throttle technique so that no task should be rejected.

Throttling Task submission rate using ThreadPoolExecutor and Semaphore

In this solution, we will create a `Semaphore` with a number which must be equal to maximum number of tasks in blocking queue at any given point of time. So the approach works like this:

1) Before executing a task a lock in semaphore is requested
2) If lock is acquired then execution works normally; Otherwise retry will happen until lock is acquired
3) Once task is completed; lock is released to semaphore

Our new throttling enabled `BlockingThreadPoolExecutor` looks like below:

```
package threadpoolDemo;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.RejectedExecutionException;
import java.util.concurrent.Semaphore;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

public class BlockingThreadPoolExecutor extends ThreadPoolExecutor
{
   private final Semaphore semaphore;

   public BlockingThreadPoolExecutor(int corePoolSize, int maximumPoolSize, lo
   {
      super(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue);
      semaphore = new Semaphore(corePoolSize + 50);
   }

   @Override
   protected void beforeExecute(Thread t, Runnable r)
   {
      super.beforeExecute(t, r);
```

```
    }

    @Override
    public void execute(final Runnable task)
    {
       boolean acquired = false;
       do
       {
          try
          {
             semaphore.acquire();
             acquired = true;
          } catch (final InterruptedException e)
          {
             //LOGGER.warn("InterruptedException whilst aquiring semaphore", e)
          }
       } while (!acquired);
       try
       {
          super.execute(task);
       } catch (final RejectedExecutionException e)
       {
          System.out.println("Task Rejected");
          semaphore.release();
          throw e;
       }
    }

    @Override
    protected void afterExecute(Runnable r, Throwable t)
    {
       super.afterExecute(r, t);
       if (t != null)
       {
          t.printStackTrace();
       }
       semaphore.release();
    }
}
```

Now test the code as below.

```
package threadpoolDemo;

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;

```

```
9  public class DemoExecutor
10 {
11    public static void main(String[] args)
12    {
13       Integer threadCounter = 0;
14       BlockingQueue<Runnable> blockingQueue = new ArrayBlockingQueue<Runnable>
15       BlockingThreadPoolExecutor executor = new BlockingThreadPoolExecutor(10,
16       executor.setRejectedExecutionHandler(new RejectedExecutionHandler()
17          {
18             @Override
19             public void rejectedExecution(Runnable r, ThreadPoolExecutor execu
20             {
21                System.out.println("DemoTask Rejected : " + ((DemoTask) r).getN
22                try
23                {
24                   Thread.sleep(1000);
25                } catch (InterruptedException e)
26                {
27                   e.printStackTrace();
28                }
29                System.out.println("Lets add another time : " + ((DemoTask) r).
30                executor.execute(r);
31             }
32          });
33       // Let start all core threads initially
34       executor.prestartAllCoreThreads();
35       while (true)
36       {
37          threadCounter++;
38          // Adding threads one by one
39          System.out.println("Adding DemoTask : " + threadCounter);
40          executor.execute(new DemoTask(threadCounter.toString()));
41          if (threadCounter == 1000)
42             break;
43       }
44    }
45 }
```

When you run the `DemoExecutor` program using `BlockingThreadPoolExecutor` in place of `CustomThreadPoolExecutor`, you will not see any task rejected and all tasks will be executed successfully.

You can control the number of tasks executing at any time parameter passes in `Semaphore` constructor.

That's all for this post. You should read more posts on **concurrency** for better confidence.

Happy Learning !!

Related Posts:

1. **How to use BlockingQueue and ThreadPoolExecutor in java**
2. **Java executor framework tutorial and best practices**
3. **When to use CountDownLatch : Java concurrency example tutorial**
4. **Difference between "implements Runnable" and "extends Thread" in java**
5. **2 ways to execute timer tasks in spring 3**
6. **Inter-thread communication using piped streams in java**

SEMAPHORE THREADPOOLEXECUTOR

PREVIOUS POST

Create eclipse templates for faster java coding

NEXT POST

How to Increase Console Output Limit in Eclipse

4 THOUGHTS ON “THROTTLING TASK SUBMISSION RATE USING THREADPOOLEXECUTOR AND SEMAPHORE”

Ramesh

JULY 23, 2014 AT 2:41 PM

Hi

This BlockingThreadPoolExecutor , will be problem for parallel execution , for tasks by multiple threads , because each task to be submitted for execution first it needs to acqcuire the lock , this will cause drop in performance

REPLY

★ Lokesh

JULY 23, 2014 AT 4:54 PM

True. But here we are trying to limit of rejected tasks to zero. In good performance, task will be added son but they may be rejected as well.

REPLY

Prashanth V

MAY 24, 2014 AT 6:14 PM

Hi,
Where is the code for BlockingThreadPoolExecutor ? I don t see it in ur post. Pls update it. Thanks.

REPLY

★ Lokesh

MAY 24, 2014 AT 6:30 PM

My bad. Weekend hangover. Updated the post. Thanks for pointing out. Much

appreciated.

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 !! *

six × 1 =

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 3,668 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