Writing a deadlock and resolving in java

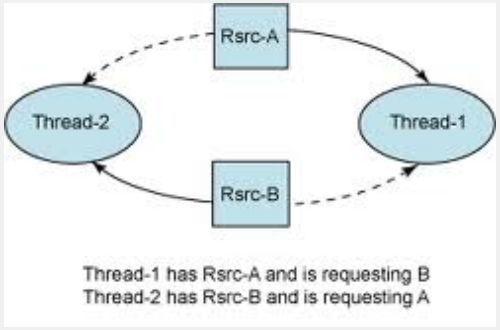
OCTOBER 16, 2012 LOKESH 5 COMMENTS

In this post, I will write a piece of code which will create a deadlock situation and then i will discuss that way to resolve this scenario.

In my previous post, i written about *Auto reload of configuration when any change happen*, i discussed about refreshing your application configuration using a thread. As configurations are shared resources and when accessing via Threads, there is always chance of writing incorrect code and caught in deadlock situation.

In java, a deadlock is a situation where minimum two threads are holding lock on some different resource, and both are waiting for other's resource to complete its task. And, none is

able to leave the lock on resource it is holding. (See image below)



In above case, Thread- has A but need B to complete processing and Similarly Thread-2 has resource B but need A first.

Let write above scenario in java code:

```
1  package thread;
2
3  public class ResolveDeadLockTest {
4
5      public static void main(String[] args) {
6          ResolveDeadLockTest test = new ResolveDeadLockTest();
7
8          final A a = test.new A();
9          final B b = test.new B();
10
11         // Thread-1
12         Runnable block1 = new Runnable() {
13             public void run() {
14                 synchronized (a) {
15                     try {
16                         // Adding delay so that both threads can start trying
17                         // lock resources
18                         Thread.sleep(100);
19                     } catch (InterruptedException e) {
20                         e.printStackTrace();
21                     }
22                     // Thread-1 have A but need B also
23                     synchronized (b) {
24                         System.out.println("In block 1");
25                     }
26                 }
27             }
28         };
```


YOU MAY ALSO LIKE

Thread synchronization, object level locking and class level locking

Difference between sleep() and wait()?

Difference between “implements

```
29
30          // Thread-2
31          Runnable block2 = new Runnable() {
32              public void run() {
33                  synchronized (b) {
34                      // Thread-2 have B but need A also
35                      synchronized (a) {
36                          System.out.println("In block 2");
37                      }
38                  }
39              }
40          };
41
42          new Thread(block1).start();
43          new Thread(block2).start();
44      }
45
46      // Resource A
47      private class A {
48          private int i = 10;
49
50          public int getI() {
51              return i;
52          }
53
54          public void setI(int i) {
55              this.i = i;
56          }
57      }
58
59      // Resource B
60      private class B {
61          private int i = 20;
62
63          public int getI() {
64              return i;
65          }
66
67          public void setI(int i) {
68              this.i = i;
69          }
70      }
71  }
```

Running above code will result in deadlock for very obvious reasons (explained above). Now we have to solve this issue.

I believe, solution to any problem lies in identifying the root of problem. In our case, it is the pattern of accessing A and B, is main issue. So, to solve it, we will simply re-order the

Runnable" and "extends Thread" in java

Java executor framework tutorial and best practices

How to use BlockingQueue and ThreadPoolExecutor in java

Why not to use finalize() method in java

statements where code is accessing shared resources.

After rewriting the code, it will look like this ::

```
    // Thread-1
Runnable block1 = new Runnable() {
    public void run() {
        synchronized (b) {
            try {
                // Adding delay so that both threads can start trying to
                // lock resources
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            // Thread-1 have A but need B also
            synchronized (a) {
                System.out.println("In block 1");
            }
        }
    }
};

// Thread-2
Runnable block2 = new Runnable() {
    public void run() {
        synchronized (b) {
            // Thread-2 have B but need A also
            synchronized (a) {
                System.out.println("In block 2");
            }
        }
    }
};
```

Run again above class, and you will not see any deadlock kind of situation. I hope, it will help you in avoiding deadlocks, and if encountered, in resolving them.
Happy Learning !!

Related Posts:

1. Thread synchronization, object level locking and class level locking
2. Difference between sleep() and wait()?
3. Difference between “implements Runnable” and “extends Thread” in java
4. Java executor framework tutorial and best practices
5. How to use BlockingQueue and ThreadPoolExecutor in java
6. Why not to use finalize() method in java

DEADLOCK JAVA MULTI THREADING

PREVIOUS POST

Manage system log files not to exceed N GB in linux using java

NEXT POST

Understanding abstraction in java

5 THOUGHTS ON “WRITING A DEADLOCK AND RESOLVING IN JAVA”

simhachalam

OCTOBER 2, 2014 AT 10:14 PM

hi LOKESH,
can i do the deadlock by using synchronized methods(not synchronized blocks).

REPLY

Ram

FEBRUARY 6, 2014 AT 6:42 AM

Hi Lokesh,
i m getting same output in both case please do need fully.

REPLY

★ Lokesh

FEBRUARY 6, 2014 AT 7:43 AM

It's not the output that matters, it's deadlock concept in discussion. Output may/or may not differ, both are perfectly fine.

REPLY

Mudassir Shahzad

JANUARY 9, 2014 AT 10:22 AM

Loved your approach of getting to the root problem by first understanding the pattern. Youd rocked it!

REPLY

Pingback: How to use BlockingQueue and ThreadPoolExecutor 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 !! *

nine − 9 =

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 3,665 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