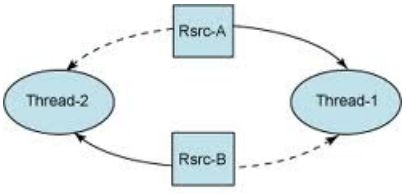
How To Do In Java

Writing a deadlock and resolving in java

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)



Thread-1 has Rsrc-A and is requesting B
Thread-2 has Rsrc-B and is requesting A

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) {
```

```
                    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 (b) {
                        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");
                    }
                }
            }
        };

        new Thread(block1).start();
        new Thread(block2).start();
    }

    // Resource A
    private class A {
        private int i = 10;

        public int getI() {
            return i;
        }

        public void setI(int i) {
            this.i = i;
        }
    }

    // Resource B
    private class B {
        private int i = 20;

        public int getI() {
            return i;
        }

        public void setI(int i) {
            this.i = i;
        }
    }
}
```

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 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");
            }
        }
    }
};
```

```
19
20  // Thread-2
21  Runnable block2 = new Runnable() {
22      public void run() {
23          synchronized (b) {
24              // Thread-2 have B but need A also
25              synchronized (a) {
26                  System.out.println("In block 2");
27              }
28          }
29      }
30  };
```

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**
7. **Automatic resource management with try-with-resources in java 7**
8. **When to use CountDownLatch : Java concurrency example tutorial**

DEADLOCK JAVA MULTI THREADING

6 THOUGHTS ON "WRITING A DEADLOCK AND RESOLVING IN JAVA"

Souvik

JANUARY 25, 2015 AT 9:42 PM

Nice example. Thanks for sharing.

simhachalam

OCTOBER 2, 2014 AT 10:14 PM

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

Ram

FEBRUARY 6, 2014 AT 6:42 AM