Java Concurrency

Kishanthan Thangarajah

Senior Software Engineer

WSO2

lean • enterprise • middleware

Agenda

- Why Concurrency?
- Processes and Threads
- Java Thread Example
- Thread Safety
- Race Conditions & Critical Sections
- Java Synchronization
- Deadlock, Starvation
- Java Concurrent APIs

Why Concurrency?

- Benefits
  - Make use of multi processor system
  - Handle asynchronous behaviour (eg : Server)
  - Better responsive applications (eg : UI)

Why Concurrency?

- Risks
  - Thread safety
  - Deadlocks, starvation
  - Performance overhead

Processes and Threads

- Process
  - Runs independently of other processes and has separate memory space.

- Thread
  - Also runs independently of other threads, but can access shared data of other threads in the same process.

- A Java application at least has one thread (main)

Java Thread Example

- Thread subclass

```java
public class ExampleThread extends Thread {
   @Override
   public void run() {
      System.out.println("Hello !!!");
   }
}



ExampleThread thread = new ExampleThread();
thread.start();
```

Java Thread Example

- Implement “Runnable” interface

```java
public class ExampleRunnable implements Runnable {
  @Override
  public void run() {
    System.out.println("Hello !!!");
  }
}


Thread thread = new Thread(new ExampleRunnable());
thread.start();
```

Java Thread Example

- Common mistake with threads

Calling **run()** instead of **start()**

This will not start a new thread, instead the run() method will be executed by the same thread.

Java Thread Example

- Pausing a thread

```
Thread.sleep(5000);
```

Example : Cluster initialization

- Interrupting a thread

```
ExampleThread thread = new ExampleThread();
thread.start();
thread.interrupt();
```

Java Thread Example

othread.join() : wait on another thread for completion

The **join** method allows one thread to wait for the completion of another.

Java Thread Example

oThread Local

o used with variables that can only be accessed (read and write) by the same thread.

```
public class ExampleThreadLocal {
   public static class ExampleRunnable implements Runnable {
      private ThreadLocal<Integer> threadLocal = new ThreadLocal<Integer>();

      @Override
      public void run() {
         threadLocal.set((int) (Math.random() * 500));
         System.out.println("Thread Local Variable Value : " + threadLocal.get());
      }
   }

   public static void main(String[] args) {
      ExampleRunnable runnable = new ExampleRunnable();
      Thread t1 = new Thread(runnable);
      Thread t2 = new Thread(runnable);
      t1.start();
      t2.start();
   }
}
```

Java Thread Example

- oThread Local
  - o Practical example : CarbonContext

- oThread Signalling
  - o wait(), notify() and notifyAll()

Thread Safety

- If multiple threads access (read or write) the same variable (or a code section) without proper synchronization, the application is not thread safe.

  - Don't share mutable variable between threads or make the variable immutable.
  - Synchronize the access of the variable.

Race Condition & Critical Sections

- If two threads try to compete for same resource and if the order in which the resource is accessed is of interest, then there arise a race condition.

- A resource (code section) that leads to race conditions is called a critical section.

Race Condition & Critical Sections

```java
public class Example {
    private int x = 0;

    public void increment() {
        x++;
    }
}
```

Race Condition & Critical Sections

- Single expression composed of multiple steps
  - Thread Interference

- Inconsistence view of the value
  - Memory consistency error

Java Synchronization

```
public class Example {
   private int x = 0;

   public synchronized void increment() {
       x++;
   }

   public synchronized int getValue() {
       return x;
   }
}
```

Java Synchronization

- No two threads can execute synchronized methods on the same object instance.
  - Locks on objects.

- Changes to the object within synchronized section are visible to all threads.
  - Resumed threads will see the updated value

Java Synchronization

- The synchronized keyword can be used with the following:
  - Instance methods or code segment blocks within instance methods
  - Static methods or code segment blocks within static methods

Java Synchronization

- Synchronized statements

```
public void increment() {
   synchronized (this) {
      x++;
   }
}
```

Java Synchronization

- Use of “static”

```
public class Example {
   public static synchronized void sayHello1() {
      System.out.println("Hello1 !!!");
   }

   public static void sayHello2() {
      synchronized (Example.class) {
         System.out.println("Hello2 !!!");
      }
   }
}
```

Java Synchronization

- Reentrant Synchronization

A thread can acquire lock already owned by it self.

```
public class Example {
    public synchronized void sayHello1() {
        System.out.println("Hello1 !!!");
        sayHello2();
    }

    public void sayHello2() {
        synchronized (this) {
            System.out.println("Hello2 !!!");
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                System.out.println("I was interrupted !!!");
            }
        }
    }
}
```

Deadlock & Starvation

- Deadlock
  - Two or more threads are blocked forever, waiting for each other.
  - Occur when multiple threads need the same locks, at the same time, but obtain them in different order.

    Thread 1 locks A, waits for B, Thread 2 locks B, waits for A
  - Practical Example : Issue found with CarbonDeploymentSchedulerTask and ClusterMessage

- Starvation
  - A thread is not given regular access to CPU time (shared resources) because of other threads.

Deadlock & Starvation

- Deadlock Prevention
  - Order how the locks can be acquired
  - Use locks instead of synchronized statements (fairness)

Java Concurrent APIs

- Found under java.util.concurrent
- High level concurrency objects
  - Locks
  - Executors
  - Concurrent Collections
  - Atomic variables

Java Concurrent APIs

- Locks

```
Lock lock = ……
lock.lock();
try {
   // critical section
} finally {
   lock.unlock();
}
```

Java Concurrent APIs

- ReentrantLock
  - Provide reentrant behaviour, same as with synchronized blocks, but with extended features (fairness).

```java
public class LockExample {
   private Lock lock = new ReentrantLock();
   private int x = 0;

   public void increment() {
       lock.lock();
       try {
           x++;
       } finally {
           lock.unlock();
       }
   }
}
```

Java Concurrent APIs

- Read/Write Locks
  - Used with the scenario where multiple readers present with only one writer.
  - Keeps a pair of locks, one for “read-only” operations and one for writing
  - Practical Example : TenantAxisUtils -> reading vs terminating tenant axisConfigurations.

Java Concurrent APIs

- Executors
  - Thread creation and management itself is a separate task when it comes to large scale applications.
  - Three main categories
    - Executor Service
    - Thread Pools
    - Fork/Join

Java Concurrent APIs

- Executors Interfaces
  1. **ExecutorService**, help manage lifecycle of the individual tasks.
  2. **ScheduledExecutorService**, supports periodic execution of tasks.
- Practical example : CarbonDeploymentSchdularTask

Java Concurrent APIs

- Thread Pool
  - Manage a pool of worker threads.
  - Reduces the overhead due to new thread creation.
  - Create thread pools using the factory methods of **java.util.concurrent.Executors**.
  - Example : Tomcat Listener Thread Pool, Synapse Worker Thread pool

Java Concurrent APIs

- Fork/Join
  - From Java 7 onwards.
  - Helps to take advantage of multi-processor system.
  - Break a large task into small tasks and execute them parallelly
  - Example : Count total number of prime numbers between 1 to 1000.
  - java.util.streams package uses implementation of fork/join framework.

Java Concurrent APIs

- Concurrent Collections
  - Help avoid memory consistency errors
  - ConcurrentMap
    - Subinterface of Map with atomic map operations.
    - Practical Example : Axis2 Deployer Map (uses ConcurrentHashMap)

  - Blocking Queue
    - FIFO data structure that blocks or times out when adding to a full queue, or get from an empty queue

Java Concurrent APIs

- Atomic Variables
  - Supports atomic operations on variables
  - Practical Example : ClusterMessage on Repository Update.

```java
public class Example {
   private AtomicInteger x = new AtomicInteger(0);

   public void increment() {
       x.incrementAndGet();
   }

   public int getValue() {
       return x.get();
   }
}
```

Questions ?