6. Многопоточность

Программирование на Java

Федор Лаврентьев

МФТИ, 2016

Потоки

Процесс и поток

- Процесс
  - Выделенная память
  - Процессорное время расходуется потоками
  - Порождается через fork
  - Общается с другими процессами через сокеты и shared memory
- Поток
  - Общая память процесса
  - Собственный кеш
  - Расходует процессорное время
  - Порождается внутри процесса
  - Общается с другими потоками через общую память процесса

java.lang.Thread

- start(), ~~run()~~
- static sleep(), static yield()
- ~~stop(), destroy()~~
- interrupt(), isInterrupted(), static interrupted()
- setDaemon(), isDaemon()
- join()
- getState():
  - NEW
  - RUNNABLE
  - BLOCKED
  - WAITING, TIMED_WAITING
  - TERMINATED

Старт потока через override

```
public void runOverridenThread() {
  Thread thread = new Thread() {
    @Override
    public void run() {
      // do something
    }
  };
  thread.start();
}
```

Старт потока с Runnable

```
public void runRunnableInThread() {
  Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
      // do something
    }
  });
  thread.start();
}
```

Прерывание потока

```
public void interruptThread() throws Exception {
  Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
      while (!Thread.interrupted()) {
        // Do something useful
      }
      // Thread interrupted
    }
  });
  thread.start();
  Thread.sleep(1);
  thread.interrupt();
}
```

Прерывание блокирующего вызова

```
public void interruptThread() throws Exception {
  Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
      while (!Thread.interrupted()) {
        // Do something useful
        try {
          Thread.sleep(1000); // blocking call
        } catch (InterruptedException e) {
          // Thread interrupted
        }
      }
      // Thread interrupted
    }
  });
  thread.start(); Thread.sleep(1); thread.interrupt();
}
```

Работа с памятью

synchronized

- Критическая секция
- Допускается повторное вхождение (reentrant)
- Синхронизироваться можно по любому объекту

- Необходимо минимизировать размер критических секций
- По возможности, дробить критические секции на части
- Можно использовать альтернативные блокировки

synchronized

```
public static synchronized void sMethod() {
  // тело метода
}

public static void sMethodDescribed () {
  synchronized (SynchronizedExample.class) {
    // тело метода
  }
}

public synchronized void iMethod() {
  // тело метода
}

public void iMethodDescribed() {
  synchronized (this) {
    // тело метода
  }
}
```

```
private final Object lock = new Object();
public void anotherObject () {
  synchronized (lock) {
    // тело метода
  }
}
```

volatile

- Форсирует работу с памятью в обход кешей
- Гарантирует атомарное чтение и запись для double и long
- Не гарантирует атомарный инкремент

private volatile long value;

Atomic*

- Базируются на операции compareAndSet(previous, next)
- Это отдельная инструкция байт-кода JVM
- Гарантирует атомарность чтения, записи и инкремента

- java.util.concurrent.atomic.*:
  - AtomicInteger
  - AtomicLong
  - AtomicBoolean
  - AtomicReference<T>
  - AtomicIntegerArray
  - AtomicLongArray

Atomic incrementAndGet()

```
public class AtomicInteger extends Number {
  private volatile int value;

  private native boolean compareAndSet(int current, int next);

  public final int incrementAndGet () {
    for (;;) {
      int current = get();
      int next = current + 1;
      if (compareAndSet(current, next)) {
        return next;
      }
    }
  }
}
```

Atomic getAndIncrement()

```
public class AtomicInteger extends Number {
  private volatile int value;

  private native boolean compareAndSet(int current, int next);

  public final int getAndIncrement() {
    for (;;) {
      int current = get();
      int next = current + 1;
      if (compareAndSet(current, next)) {
        return current;
      }
    }
  }
}
```

Immutable

- Неизменяемые объекты не нуждаются в синхронизации состояния, а значит, потокобезопасны

- Все поля – final (либо не изменяются)
- Все mutable параметры конструктора скопированы
- Ссылки на внутренние mutable объекты не публикуются
- Родительский класс также immutable
- Нет утечки ссылки на this из конструктора

Проблемы многопоточности

Deadlock

```
class Account {
  public int money;
}


class AccountManager {
  public void transfer(Account from, Account to, int amount) {
    assert amount > 0;


    synchronized (from) {
      assert from.money >= amount;
      synchronized (to) {
        to.money += amount;
        from.money -= amount;
      }
    }
  }
}
```

Lock ordering

```
class Account {
  public int money;
  public final long id;
}

class AccountManager {
  public void transfer(Account from, Account to, int amount) {
    assert amount > 0;
    assert from.number != to.number;
    Object first = from.number < to.number ? from : to;
    Object second = from.number < to.number ? to : from;
    synchronized (first) {
      synchronized (second) {
        to.money += amount;
        from.money -= amount;
      }
    }
  }
}
```

Livelock

- Поток успешно получает блокировку, однако выполнить полезное действие не может
- Два потока конфликтуют, откатывают результаты своей работы и пытаются через фиксированное время повторить попытку (например, делают sleep(1000)).
- При следующей попытке они опять будут конфликтовать
- Пример – задача про обедающих философов
- Решение – случайная задержка

Starvation

- Бесконечные циклы – холостая утилизация CPU
- sleep при взятой блокировке
- Blocking IO при взятой блокировке
- Неоптимальные приоритеты потоков в ОС

Решение проблем

- Не использовать многопоточность
- Не разделять состояние между потоками
- Разделять только immutable-состояние
- Использовать lock-free алгоритмы
- Использовать готовые примитивы синхронизации
- Аккуратнее программировать синхронизацию

Примитивы синхронизации

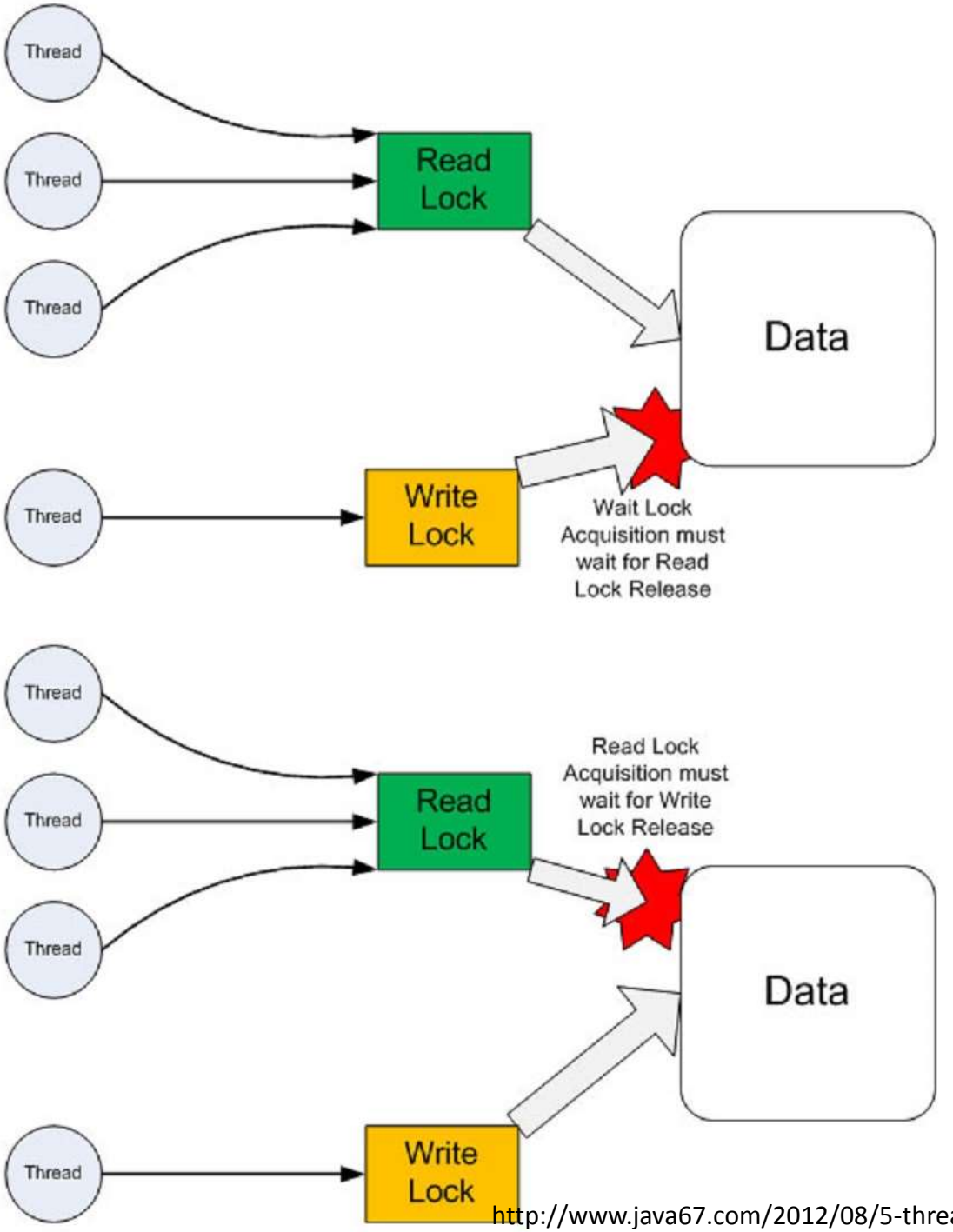
https://habrahabr.ru/company/luxoft/blog/157273/

Lock (ReentrantLock)

- lock()
- unlock()
- tryLock()
- tryLock(long timeout)

ReadWriteLock

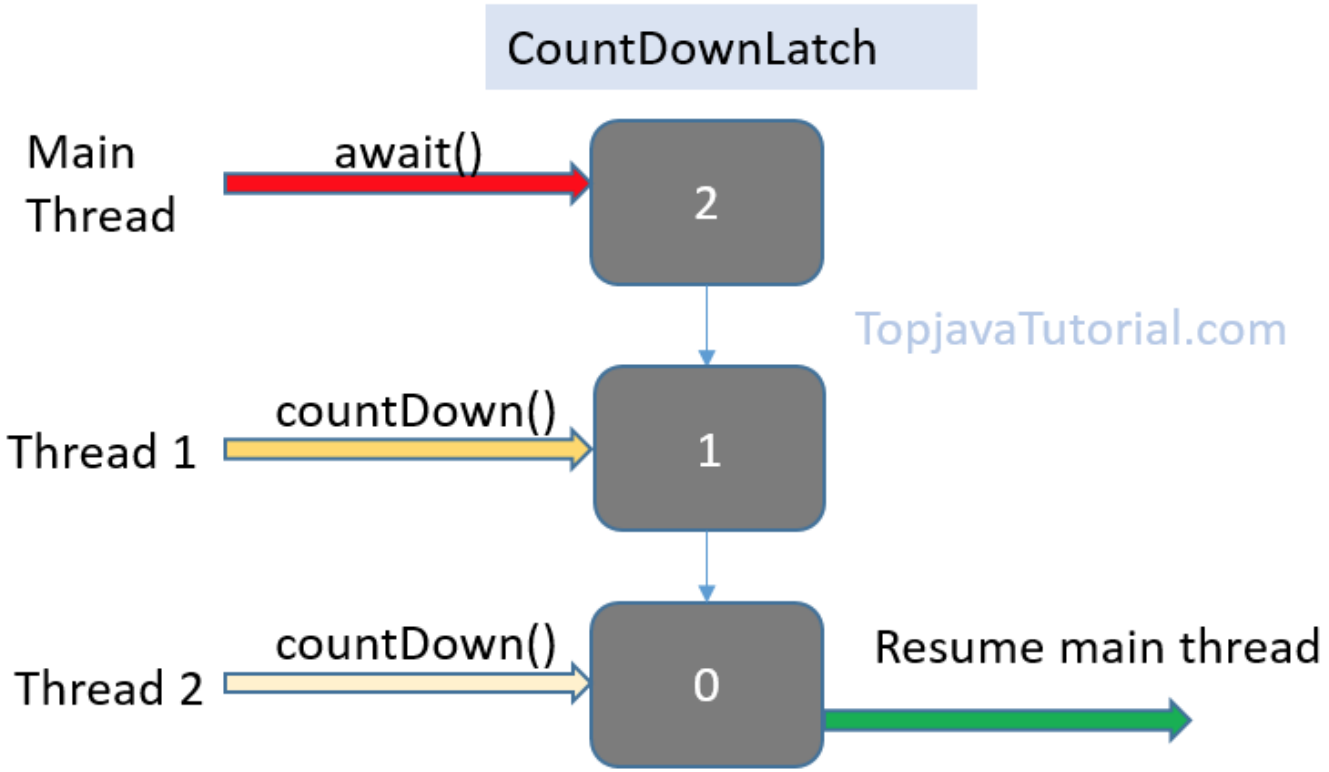
- readLock().lock()
- readLock().unlock()
- writeLock().lock()
- writeLock().unlock()



http://www.java67.com/2012/08/5-thread-interview-questions-answers-in.html

CountDownLatch

- new CountDownLatch(int count)
- await()
- countDown()



http://www.topjavatutorial.com/java/countdownlatch-in-java/

А также

- Semaphore
- CyclicBarrier
- Exchanger
- Phaser

Concurrent-коллекции

- Vector, Stack, Hashtable
- Collections.synchronized*()
- ConcurrentHashMap, ConcurrentSkipListMap
- CopyOnWriteArrayList, CopyOnWriteArraySet
- BlockingQueue – ArrayBQ, LinkedBQ
- ConcurrentLinkedQueue