Лекция 5.4: Многопоточность / synchronized, wait/notify.

Ваша успеваемость

Java Core	14.34%
1. Основы Java	88.07%
2. Базовые алгоритмы	24.13%
3. Исключения	31.21%
4. Ввод/вывод	0%
5. Многопоточность	0%
6. Коллекции	0%
7. ООП: Синтаксис	0%
8. ООП: Шаблоны	0%
9. Продвинутые возможности	0%
10. Java 8	0%

Тест

Для прохождения теста по теме XX_thread.monitor-1refs.rules нажмите “Start Quiz”

Тест

Для прохождения теста по теме XX_thread.monitor.waitset-blockedset нажмите “Start Quiz”

Тест

Для прохождения теста по теме XX_thread.monitor-2refs.rules нажмите “Start Quiz”

Тест

Для прохождения теста по теме XX_thread.monitor-static.simultaneously нажмите “Start Quiz”

Тест

Для прохождения теста по теме XX_thread.monitor-static-equal нажмите “Start Quiz”

1. Ключевое слово synchronized

1.1 Модификатор метода synchronized / synchronized-секция

Существует две формы применения synchronized – модификатор метода или synchronized-блок.
Вот пример с модификатором метода

```java
public class App {
    public synchronized void f() {}
}
```

Вот пример с synchronized-блоком (блоку всегда нужна ненулевая ссылка на произвольный объект)

```java
public class App {
    public void f() {
        Object ref = new Object();
        synchronized(ref) {}
    }
}
```

Они эквивалентны с точки зрения многопоточности: синхронизированный метод экземпляра эквивалентен секции синхронизации по this схватывающей весь код метода

```java
public class App {
    public synchronized void f() {
        // тело метода
    }
    public void g() {
        synchronized(this) {
            // тело метода
        }
    }
}
```

Синхронизированный статический метод эквивалентен секции синхронизации по экземпляру класса

```java
public class App {
    public synchronized static void f() {
        // тело метода
    }
    public static void g() {
        synchronized(App.class) {
            // тело метода
        }
    }
}
```

Порядок вызова synchronized-метода эквивалентен вызову обычного метода – вычисление аргументов слева-направо, потом вход. Преимущество секции синхронизации в том, что Вы сами определяете границы синхронизации и то, по чему синхронизироваться. Например можно сделать вот так

```java
public class App {
    public void g(int[] arg0, Integer arg2) {
        System.out.println("A");
        synchronized(arg0) {
            System.out.println("B");
            synchronized(arg1) {
                System.out.println("C");
            }
            System.out.println("D");
        }
        System.out.println("E");
        synchronized(arg1) {
            System.out.println("F");
        }
        System.out.println("G");
    }
}
```

Попытка синхронизации на null закончится NPE

```java
public class App {
    public static void g(Object ref) {
        Object ref = null;
        synchronized(ref) {}
    }
}

>> ???
```

1.2 Свойства synchronized (reentrancy, не коррелирует с другими)

synchronized обладает свойством reentrancy, т.е. имеет возможность повторного входа одним и тем же потоком внутрь секции синхронизации по одной и той же переменной (отметим, что этим свойство обладают не все блокировки, скажем ReentrantLock – обладает, а StampedLock – нет ???).

```java
public class App {
    public static void g(Object ref) {
        Object ref = new Object();
        System.out.println("Я снаружи");
        synchronized(ref) {
            System.out.println("  Я внутри");
            synchronized(ref) {
                System.out.println("    Я дважды внутри!");
            }
            System.out.println("  Опять внутри");
        }
        System.out.println("Опять снаружи");
    }
}

>> Я снаружи
>>   Я внутри
>>     Я дважды внутри!
>>   Опять внутри
>> Опять снаружи
```

Ситуация с повторным входом часто возникает при вызове одного synchronized-метода из другого (для гурманов – рекурсия синхронизированного метода)

```java
public class App {
    public static void main(String[] args) {
        f();
    }
    public static synchronized void f() {
        g();
    }
    public static synchronized void g() {
        h();
    }
    public static synchronized void h() {
    }
}
```

Узнать захватил находится ли наш поток в synchronized по данной ссылке можно при помощи метода Thread.holdsLock(…)

```java
public class App {
    public static void main(String[] args) {
        Object ref = new Object();
        System.out.println(Thread.holdsLock(ref));
        synchronized (ref) {
            System.out.println(Thread.holdsLock(ref));
        }
    }
}

>> false
>> true
```

```java
public class App {
    public static void main(String[] args) {
        System.out.println(Thread.holdsLock(App.class));
        f();
    }
    public static synchronized void f() {
        System.out.println(Thread.holdsLock(App.class));
    }
}

>> false
>> true
```

1.3 Статический synchronized-метод

1.4 happend-before по захват/освобождение

2. Взаимное исключение + взаимная блокировка

3. Полностью синхронизированные объекты

3.1 История с Vector, Hashtable, StringBuffer, ???

3.2 Современные оптимизации synchronized в HotSpot

3.3 Опасность открытости, шаблон Private Mutex

4. Концепция монитора: synchronized + wait()/notify()/notifyAll()

5. Пример использования: ограниченная блокирующая очередь

Рассмотрим следующий код

```java
public class SingleElementBuffer {
    private Integer elem = null;

    public synchronized void put(int newElem) throws InterruptedException {
        while (this.elem != null) {
            this.wait();
        }
        this.elem = newElem;
        this.notifyAll();
    }

    public synchronized int get() throws InterruptedException {
        while (elem == null) {
            this.wait();
        }
        Integer result = this.elem;
        this.elem = null;
        this.notifyAll();
        return result;
    }
}
```

Или аналогичный по функционалу но с использованием идиомы Private Mutex

```java
public class SingleElementBuffer {
    private final Object lock = new Object();
    private Integer elem = null;

    public void put(int newElem) throws InterruptedException {
        synchronized (lock) {
            while (this.elem != null) {
                lock.wait();
            }
            this.elem = newElem;
            lock.notifyAll();
        }
    }

    public synchronized int get() throws InterruptedException {
        synchronized (lock) {
            while (elem == null) {
                lock.wait();
            }
            Integer result = this.elem;
            this.elem = null;
            lock.notifyAll();
            return result;
        }
    }
}
```

```
???
```

```
???
```

```
???
```

6. Критика synchronized

6.1 Не “честный” (non fair)

6.2 Нет возможности проверить захвачен ли

6.3 Нет возможности попробовать захватить с timeout

6.4 Нет возможности узнать размер blocked-set

6.5 Всего одна условная переменная

6.6 Не учитывает “приоритеты” потоков

7. Детали реализации

7.1 На уровне байткода

7.2 На уровне JVM/HotSpot

шаблон Private Mutex: внешняя блокировка, кража сигналов, синхронизация по строке

Есть две синтаксические формы основанные на synchronized:
- synchronized метод (экземпляра(не статический) или класса(статический))
- synchronized блок (по произвольной не нулевой ссылке)

Нельзя ???маркать при помощи synchronized
- классы
- поля
- локальные переменные
- методы интерфейса

synchronized keyword не входит в сигнатуру метода. И, следовательно, synchronized автоматически не наследуется когда подклассы переопределяют (override) метод предка и методы интерфейса не могут быть отмечены synchronized.
Также конструкторы не могут быть synchronized, хотя в конструкторах может быть использован synchronized-блок.

Тесты по всей лекции

Тест, состоящий из случайных вопросов тестов этой лекции

Some issues occured with quiz randomizer :(

Видео

Набор декабрь 2013
Набор июль 2013
Набор апрель 2013
Набор февраль 2013
Набор октябрь 2012

Литература

Простая

Шилд. Java. Полное руководство. Издание 8
….Глава 11. Многопоточное программирование, стр 272-276

Хорстманн, Корнелл. “Java 2. Том II. Тонкости программирования”. Издание 7
….Глава 1. Многопоточность, стр 47-70

Bill Venners, “Thread Synchronization”

Средняя
- Глава 2.2 Synchronization из “Doug Lea. Concurrent Programming in Java. 2ed”.

Сложная
- Как реализована функциональность монитора в HotSpot 7 = “Synchronization and Object Locking”