Проблема многопоточности — локальный кэш. Volatile

Java Core (/quests/QUEST_JAVA_CORE)
7 уровень (/quests/lectures/?quest=QUEST_JAVA_CORE&level=7), 5 лекция (/quests/lectures/questcore.level07.lecture05)

ОТКРЫТА

— Привет, Амиго! Помнишь, Элли тебе рассказывала про проблемы при одновременном доступе нескольких нитей к общему (разделяемому) ресурсу?

— Да.

— Так вот – это еще не все. Есть еще небольшая проблема.

Как ты знаешь, в компьютере есть память, где хранятся данные и команды (код), а также процессор, который исполняет эти команды и работает с данными. Процессор считывает данные из памяти, изменяет и записывает их обратно в память. Чтобы ускорить работу процессора в него встроили свою «быструю» память – кэш.

Чтобы ускорить свою работу, процессор копирует самые часто используемые переменные и области памяти в свой кэш и все изменения с ними производит в этой быстрой памяти. А после – копирует обратно в «медленную» память. Медленная память все это время содержит старые(!) (неизмененные) значения переменных.

И тогда может возникнуть проблема. Одна нить меняет переменную, такую как isCancel или isInterrupted из примера выше, а вторая нить «не видит» этого изменения, т.к. оно было совершено в быстрой памяти. Это следствие того, что нити не имеют доступа к кэшу друг друга. (Процессор часто содержит несколько независимых ядер и нити физически могут исполняться на разных ядрах.)

Вспомним вчерашний пример:

Код

Описание

```
class Clock implements Runnable
{
private boolean isCancel = false;

public void cancel()
{
this.isCancel = true;
}

public void run()
{
while (!this.isCancel)
{
Thread.sleep(1000);
System.out.println("Tik");
}
}
}
```

Нить «не знает» о существовании других нитей.

В методе run переменная isCancel при первом использовании будет помещена в кэш дочерней нити. Эта операция эквивалентна коду:

```
public void run()
{
boolean isCancelCached = this.isCancel;
while (!isCancelCached)
{
Thread.sleep(1000);
System.out.println("Tik");
}
}
```

Вызов метода cancel из другой нити поменяет значение переменной isCancel в обычной (медленной) памяти, но не в кэше остальных нитей.

```
public static void main(String[] args)
{
Clock clock = new Clock();
Thread clockThread = new Thread(clock);
clockThread.start();

Thread.sleep(10000);
clock.cancel();
}
```

— Ничего себе! А для этой проблемы тоже придумали красивое решение, как в случае с synchronized?

— Ты не поверишь!

Сначала думали отключить работу с кэшем, но потом оказалось, что из-за этого программы работают в разы медленнее. Тогда придумали другое решение.

Было придумано специальное ключевое слово volatile. Помещение его перед определением переменной запрещало помещать ее значение в кэш. Вернее не запрещало помещать в кэш, а просто принудительно всегда читало и писало ее только в обычную (медленную) память.

Вот как нужно исправить наше решение, чтобы все стало отлично работать:

Код

Описание

```
1  class Clock implements Runnable
2  {
3  private volatile boolean isCancel = false;
4
5  public void cancel()
6  {
7  this.isCancel = true;
8  }
9
10 public void run()
11 {
12 while (!this.isCancel)
13 {
14 Thread.sleep(1000);
15 System.out.println("Tik");
16 }
17 }
18 }
```

Из-за модификатора volatile чтение и запись значения переменной всегда будут происходить в обычной, общей для всех нитей, памяти.

```
1  public static void main(String[] args)
2  {
3  Clock clock = new Clock();
4  Thread clockThread = new Thread(clock);
5  clockThread.start();
6
7  Thread.sleep(10000);
8  clock.cancel();
9  }
```

— И все?

— Да. Просто и красиво.

< (/quests/lectures/questcore.level07.lecture04)

> (/quests/lectures/questcore.level07.lecture06)

Комментарии (45) популярные новые старые

Никита

Введите текст комментария

Atom 17 уровень позавчера, 16:53

Зачем в этом примере volatile, если переменная не статическая?
Все равно каждая нить себе создаст свою переменную.

Ответить 0

Vaiki 18 уровень, Минск 18 марта, 20:32

а массивов это не касается? недавно ведь в задаче без синхронизации добавляли в массив текущий индекс и все нити видели изменения, никто из них не начал параллельно в кэше с 0,1,2,n... и при синхронизации одна нить второй нити показывает, что она изменила, значит в "медленной " памяти все происходит?

Ответить 0

Алексей Кузнецов 21 уровень, Новосибирск 6 марта, 19:53

T.e. при многопоточности volatile должен у общих переменных присутствовать всегда по умолчанию? Иначе будет ая-яй-яй. Складывается ощущение, что в java очень коряво реализована многопоточность и работа с ней, видимо, в угоду кроссплатформенности.

Ответить +1

Дмитрий 31 уровень, Москва 15 января, 16:13

Протестировал скорость работы с volatile переменными - скорость ~ в 20-25 раз ниже.

Ответить +3

Сергей 30 уровень 5 января, 22:23

из лекции
"Одна нить меняет переменную, такую как isCancel или isInterrupted из примера выше, "
выше ничего нет. тот самый пример на самом деле ниже.

Ответить 0

Alex 30 уровень 10 января, 16:44

главное что нашел

Ответить +2

MaKaRi4 28 уровень, Киев 3 января, 18:50

Народ, оставляйте пожалуйста свою благодарность Максиму под его комментарием! Зачем флудить?

Ответить +3

rotarru 20 уровень, Минск 1 февраля, 23:06

☺

Плывут пароходы — привет Максиму!
Пролетают летчики — привет Максиму!
Пробегут паровозы — привет Максиму!
А пройдут пионеры — салют Максиму!

Ответить +3

Александр 18 уровень, Киев 18 декабря 2017, 02:11

Пару видео по теме:

Многопоточность: Volatile

Synchronized vs Volatile

Ответить +6

Vasya Hurmach 20 уровень, Киев 10 декабря 2017, 12:02

Так понимаю все зависит от компьютера (просто перестраховка в случае у пользователя, слабая машина). У меня все работает нормально.

Ответить 0

MrDudec 22 уровень 7 ноября 2017, 20:32

15 лекций назад было сказано, что многотиевость называть многопоточностью - ошибка, а теперь лекция о проблеме многонитиевости называется "Проблема многопоточности — локальный кэш. Volatile".

Ответить +3

Денис 26 уровень, Москва 7 февраля, 13:35

я так понимаю если как я изучать джаву с нуля, то слово нить и многонитиевость не режет ухо. А вот те кто давно в джаве, не могут слово многонитиевость переносить))) так что нам рассказали, что нить это правильное название, но привыкать надо к слову многопоточность. Иначе в будущем коллективе коллеги будут косо смотреть)))) на GeekBrains на уроке когда проходили многопоточность препод тоже сказал, что правильно нить говорить. Но в чате масса учеников написала, что режет слух)))

Ответить 0

meshuggah 17 уровень, Москва 24 октября 2017, 20:20

Насколько я понял, в примере, где переменная isCancel еще не volatile, подразумевается, что дочерняя нить не остановит свою работу, т.к. не знает, что другой поток изменил эту переменную и "тиков" будет больше 10. Я тупо скопировал код в идею и тиков было ровно 10... Что-то не сходится...

Ответить 0

MrDudec 22 уровень 7 ноября 2017, 20:30

Подразумевается, что сложные программы из-за этого могут работать не так как задумано, а это просто пример.

Ответить 0

Загрузить еще