Кейс №2

🔑 Объяснение

Периодическая задача по пересчёту рисков всегда захватывает блокировку `risksLock` на запись, так как предполагает изменение рисков. Вместе с тем, финальным шагом пересчёта является очистка кэша `visitCache`; это нужно потому, что рекомендованные ранее визиты могли утратить свою актуальность в виду изменившейся обстановки по заболеваемости.

Однако очистка данных в случае `ConcurrentHashMap` – отнюдь не атомарная операция. Внутри она содержит обход массива корзин, из которых состоит `HashMap`, и для того чтобы вычистить корзину, на ней захватывается блокировка обычной секцией `synchronized`, см. `java/util/concurrent/ConcurrentHashMap.java:1200`.

Эта попытка захвата в данном случае блокирует поток (назовём его №1) на неопределённый срок, потому что в это же самое время другой поток, выполняющий метод `fetchRecommendedVisits` (назовём его №2), уже захватил блокировку на этой же самой корзине (по `petId`) через метод `computeIfAbsent`:

```
    public List<Visit> fetchRecommendedVisits(int petId) {
        return visitCache.computeIfAbsent(petId, this::recommendVisits);
    }
```

, поэтому пока он не завершится, поток №1 не сможет продолжить свою работу.

Но поток №2 тоже не может продолжить выполнение, потому что ему для этого нужно захватить блокировку на чтение рисков:

```
    private List<Visit> recommendVisits(int petId) {
        risksLock.readLock().lock();
```

, однако он не может этого сделать, потому что этой блокировкой (только на запись) владеет поток №1.

Таким образом, поток №1 захватил блокировку на запись рисков и ждёт освобождения блокировки на корзине, которой владеет поток №2, а поток №2 не может освободить её, потому что ждёт освобождения блокировки на запись рисков, которой владеет поток №1.

Верхушка стека потока №1:

```
"scheduling-1" #269 [199221] prio=5 os_prio=0 cpu=6.90ms elapsed=15.47s
tid=0x00007ca1e24d6670 nid=199221 waiting for monitor entry
[0x00007ca1c07fa000]
   java.lang.Thread.State: BLOCKED (on object monitor)
  at
java.util.concurrent.ConcurrentHashMap.clear(java.base@21.0.3/ConcurrentHashMap
.java:1200)
  - waiting to lock <0x0000000617d4c490> (a
java.util.concurrent.ConcurrentHashMap$ReservationNode)
  at
org.springframework.samples.petclinic.service.DiseaseRiskAi.doAiMagic(DiseaseRi
skAi.java:113)
  at
org.springframework.samples.petclinic.service.DiseaseRiskAi.recalculateDiseaseR
isks(DiseaseRiskAi.java:95)
```

Верхушка стека потока №2:

```
"http-nio-9966-exec-1" #257 [199209] daemon prio=5 os_prio=0 cpu=72.42ms
elapsed=15.49s tid=0x00007ca1e24c1b30 nid=199209 waiting on condition
[0x00007ca1c02f9000]
   java.lang.Thread.State: WAITING (parking)
  at jdk.internal.misc.Unsafe.park(java.base@21.0.3/Native Method)
  - parking to wait for  <0x000000060e0646b8> (a
java.util.concurrent.locks.ReentrantReadWriteLock$NonfairSync)
  at
java.util.concurrent.locks.LockSupport.park(java.base@21.0.3/LockSupport.java:2
21)
  at
java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(java.base@21.0.3/
AbstractQueuedSynchronizer.java:754)
  at
java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireShared(java.base@2
1.0.3/AbstractQueuedSynchronizer.java:1079)
  at
java.util.concurrent.locks.ReentrantReadWriteLock$ReadLock.lock(java.base@21.0.
3/ReentrantReadWriteLock.java:738)
  at
org.springframework.samples.petclinic.service.DiseaseRiskAi.recommendVisits(Dis
easeRiskAi.java:69)
  at
org.springframework.samples.petclinic.service.DiseaseRiskAi$$Lambda/0x00007ca16
cb3c1c8.apply(Unknown Source)
  at
java.util.concurrent.ConcurrentHashMap.computeIfAbsent(java.base@21.0.3/Concurr
entHashMap.java:1708)
  - locked <0x0000000617d4c490> (a
java.util.concurrent.ConcurrentHashMap$ReservationNode)
  at
org.springframework.samples.petclinic.service.DiseaseRiskAi.fetchRecommendedVis
its(DiseaseRiskAi.java:65)
  at
org.springframework.samples.petclinic.rest.controller.PetRestController.listRec
ommendedVisits(PetRestController.java:98)
```

```
...
```

В общем случае вероятность воспроизведения этой ситуации не очень высока, однако благодаря задержке в методе `doAiMagic`, имитирующей работу искусственного интеллекта, появляется 5-секундное окно, в течение которого проблема воспроизводится стабильно.

✅ Варианты решения

- Очищать кэш визитов *после* освобождения блокировки на запись рисков.
- Очищать кэш заранее, *до* начала пересчёта рисков.
- Вместо вызова `lock()` использовать более “мягкие” методы `tryLock()` с таймаутом или без.
- Наполнять кэш другим способом, например, с ручными проверками на присутствие элементов.
- *(ваш вариант)*