Кейс №4

🔑 Объяснение

Метод автоматического обновления родословной `PedigreeService#updatePedigreeBySchedule` выполняет почти всю работу в пределах одной транзакции, границы и управление которой обеспечивает метод `doInTransaction`. Помимо работы непосредственно с БД (обновления имени питомца), в рамки этой же транзакции входит и обновление кэша родословных `pedigreeTextCache`, чтобы в случае сбоя при обновлении кэша транзакция также была откачена. Поскольку кэш построен на обёртке вокруг потоко-небезопасной структуры `HashMap`, обход его элементов защищается от модификаций дополнительной внешней синхронизацией на объекте самой обёртки (см. javadoc к методу `java.util.Collections#synchronizedMap`).

Метод ручного обновления родословной `PedigreeService#updatePedigreeByRequest` в первую очередь занимается обработкой данных в кэше, поэтому его тело почти целиком обрамлено в секцию `synchronized (pedigreeTextCache)`. В ней, помимо работы с кэшем, также осуществляется обновление имени питомца в БД, которое для унификации также оборачивается в вызов `doInTransaction`.

Принципиально важным отличием методов является порядок обновления: в первом сначала обновляется БД, затем кэш, а во втором наоборот – сначала кэш, затем БД. Помимо порядка вызова методов это проявляется в том, что у первого метода на верхнем уровне стоит обрамление в транзакцию и внутри – обрамление в `synchronized`, а у второго – наоборот. Это приводит к следующей ситуации (пункты в порядке времени):

1. Поток автоматического обновления (№1) открывает транзакцию и меняет в БД имя питомца; транзакцию не закрывает.
2. Поток ручного обновления (№2) захватывает блокировку на объекте кэша и обновляет в нём все упоминания имени питомца; блокировку держит.
3. Поток №1 пытается захватить блокировку на кэше, чтобы обновить данные в нём, но не может из-за шага 2 и замирает в ожидании.
4. Поток №2 открывает новую транзакцию и пытается обновить имя питомца, но не может и остаётся ждать, так как уровень изоляции транзакций `SERIALIZABLE` требует завершения предыдущей транзакции, поменявшей эту же строку. Больше того, именно в PostgreSQL время этого ожидания по умолчанию не ограничено.
5. Таким образом, поток №1 не может захватить блокировку на кэше, потому что её держит поток №2, а поток №2 не может её освободить, потому что ждёт завершения транзакции, начатой потоком №1.

Верхушка стека потока №1:

```
"scheduling-1" #272 [166302] prio=5 os_prio=0 cpu=50.57ms elapsed=223.58s
tid=0x000071f56e4747d0 nid=166302 waiting for monitor entry
[0x000071f541ffd000]
   java.lang.Thread.State: BLOCKED (on object monitor)
  at
org.springframework.samples.petclinic.service.PedigreeService.lambda$updatePedi
greeBySchedule$1(PedigreeService.java:100)
  - waiting to lock <0x000000060ed85500> (a
java.util.Collections$SynchronizedMap)
  at
org.springframework.samples.petclinic.service.PedigreeService$$Lambda/0x000071f
4f8aa3e38.execute(Unknown Source)
  at
org.springframework.samples.petclinic.service.PedigreeService.doInTransaction(P
edigreeService.java:136)
  at
org.springframework.samples.petclinic.service.PedigreeService.updatePedigreeByS
chedule(PedigreeService.java:88)
  ...
```

Верхушка стека потока №2:

```
"http-nio-9966-exec-2" #261 [166291] daemon prio=5 os_prio=0 cpu=6.69ms
elapsed=223.60s tid=0x000071f56e460620 nid=166291 runnable
[0x000071f5418f8000]
   java.lang.Thread.State: RUNNABLE
  at sun.nio.ch.Net.poll(java.base@21.0.3/Native Method)
...
  at java.net.Socket$SocketInputStream.read(java.base@21.0.3/Socket.java:1099)
...
  at
org.postgresql.jdbc.PgPreparedStatement.executeUpdate(PgPreparedStatement.java:
152)
  at
com.zaxxer.hikari.pool.ProxyPreparedStatement.executeUpdate(ProxyPreparedStatem
ent.java:61)
  at
com.zaxxer.hikari.pool.HikariProxyPreparedStatement.executeUpdate(HikariProxyPr
eparedStatement.java)
  at
org.springframework.samples.petclinic.service.PedigreeService.updateName(Pedigr
eeService.java:172)
  at
org.springframework.samples.petclinic.service.PedigreeService.lambda$updatePedi
greeByRequest$0(PedigreeService.java:78)
  at
org.springframework.samples.petclinic.service.PedigreeService$$Lambda/0x000071f
4f8b47c50.execute(Unknown Source)
  at
org.springframework.samples.petclinic.service.PedigreeService.doInTransaction(P
edigreeService.java:136)
  at
org.springframework.samples.petclinic.service.PedigreeService.updatePedigreeByR
equest(PedigreeService.java:78)
```

```
...
```

В общем случае вероятность стечения таких обстоятельств довольно мала, поэтому, чтобы облегчить воспроизведение проблемы, в поток №1 после первого шага (обновления записи в БД) введена задержка в 5 секунд перед попыткой захвата блокировки на кэш. Проблема воспроизводится только при вызове ручного обновления в эти 5 секунд.

✅ Варианты решения

- Выровнять поведение методов – обновлять БД и кэш в одном и том же порядке.
- Сузить границы транзакции – вынести обновление кэша из транзакции при автоматическом вызове и вынести обновление БД из секции блокировки при ручном вызове.
- Использовать другую потоко-безопасную структуру данных для кэша, например, `ConcurrentHashMap`.
- Ввести явный признак активного процесса обновления, например, через `ReentrantReadWriteLock`.
- Сменить уровень изоляции транзакций на другой, см. параметр `spring.datasource.hikari.transaction-isolation` в файле `src/main/resources/application.properties`.
- Ограничить таймаут ожидания захвата блокировки в Postgre lock_timeout, который по умолчанию не выставлен:

  > A value of zero (the default) disables the timeout.

- *(ваш вариант)*