Кейс №1

🔑 Объяснение

Адрес API Реестра, прописанный в свойстве `pet-registry-url` в `application.properties`, на самом деле ведёт на отладочный сервис [httpbin.org](), а концовка формируемого URI – на его метод `/delay`, отвечающий с заданной задержкой. Аргументом вызова метода (длиной задержки в секундах) всегда является единица, полученная делением ID питомца на самого себя.

Таким образом, для каждого питомца вызывается задержка в 1 секунду на стороне сервиса `httpbin.org`, что в случае массового наплыва запросов может исчерпать пул потоков в Tomcat и привести к частичным отказам. Суммарная задержка складывается из числа питомцев (коих в базе из коробки 13 штук) и времени задержки на каждого (1 секунда), то есть составляет примерно 13 секунд.

Примечательно, что поток-обработчик входящего HTTP-запроса из пула Tomcat не занят обменом данными с внешним сервисом, вместо этого он занят ожиданием (на первый взгляд, таким же, как и все его соседи по пулу):

```
"http-nio-9966-exec-1@13806" tid=0xb7 nid=NA waiting
  java.lang.Thread.State: WAITING
    at jdk.internal.misc.Unsafe.park(Unsafe.java:-1)
    at java.util.concurrent.locks.LockSupport.park(LockSupport.java:221)
...
    at
org.springframework.web.client.DefaultRestClient$DefaultRequestBodyUriSpec.retr
ieve(DefaultRestClient.java:439)
    at
org.springframework.samples.petclinic.service.PetRegistryService.isPetRegistere
d(PetRegistryService.java:43)
...
    at
org.springframework.samples.petclinic.rest.controller.PetRestController.listPet
s(PetRestController.java:96)
...
```

Это объясняется тем, что под капотом Spring'ового клиента `RestClient` по умолчанию работает встроенный в JDK `java.net.http.HttpClient`, который в целях оптимизации производительности отправляет запросы не из потока-инициатора, а из отдельного (единственного) потока, работающего на неблокирующем вводе-выводе:

```
HttpClient-1-SelectorManager@11792" tid=0x34 nid=NA runnable
  java.lang.Thread.State: RUNNABLE
    at sun.nio.ch.EPoll.wait(EPoll.java:-1)
    at sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:121)
    at sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:130)
    - locked <0x3c8b> (a sun.nio.ch.EPollSelectorImpl)
    - locked <0x3c8c> (a sun.nio.ch.Util$2)
    at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:142)
    at
jdk.internal.net.http.HttpClientImpl$SelectorManager.run(HttpClientImpl.java:13
66)
```

Он-то и находится в состоянии `RUNNABLE` вместо инициатора.

✅ Варианты решения

- Использовать более быстрый API метод Реестра, например, `/anything` (достаточно сменить в URI слово `delay` на `anything`).
- Применить в методе `PetRestController#listPets` параллельный стрим вместо обычного последовательного.
- Собрать все данные о регистрации животных параллельно через неблокирующий HTTP клиент (например, `WebClient` из модуля WebFlux) и использовать их для фильтрации списка питомцев перед отдачей.
- Опрашивать наличие регистрации животных отдельной фоновой задачей и хранить результаты в кэше, сверка с которым в момент выполнения запроса `GET /api/pets` будет несравнимо быстрее.
- *(ваш вариант)*