УДК 004.4

DOI: 10.31673/2412-9070.2022.013133

А. М. ТУШИЧ, доктор філософії, доцент,
Державний університет телекомунікацій, Київ

ВИКОРИСТАННЯ CIRCUIT BREAKER У ПРОГРАМНОМУ ЗАБЕЗПЕЧЕННІ З МІКРОСЕРВІСНОЮ АРХІТЕКТУРОЮ

Мікросервіси — це стиль архітектури, в якому одна програма створюється як набір невеликих програм, кожна з яких працює у своєму власному процесі та взаємодіє з іншими за допомогою простих і швидких протоколів передавання даних. Крім безлічі переваг цей спосіб проєктування архітектури має важливий недолік: якщо в системі запущено багато окремих сервісів, будь-який програмний вузол може вийти з ладу, спричинюючи каскад помилок. Щоб запобігти поширенню каскадного завершення мікросервісів, застосовують Circuit Breaker — бібліотеку відмовостійкості, що описує стратегію запобігання каскадному збою на різних рівнях програми та відстежує методи невдалих викликів відповідних служб. У статті описано схеми використання Circuit Breaker та рекомендації щодо їхньої коректної роботи.

Ключові слова: мікросервісна архітектура; проксі-сервіс; Circuit Breaker; перемикач; відмовостійкість.

ВСТУП

Аналіз предметної сфери. Постановка проблеми

Circuit Breaker увійшов у світ програмування з електроніки та електротехніки. Шаблоном проєктування систем виступає запобіжник — автоматичний вимикач. Коли електрична схема працює належним чином, цей компонент системи здатен успішно проводити струм. У разі короткого замикання вимикач розмикає електричне коло, запобігаючи поширенню проблеми по всій системі.

Якщо в системі запущено багато окремих мережних служб, будь-який програмний вузол може вийти з ладу і спричинити низку помилок (наприклад, якщо немає відповіді від бази даних, то решта служб, залежних від неї, будуть відмовляти). Тому дуже важливо запобігти поширенню помилок, коли виходить із ладу тільки один мікросервіс. Решта системи має працювати стабільно, навіть без деякого набору непрацюючих сервісів.

Тому до програмних систем ця модель почала застосовуватись через поширення архітектури мікросервісів. Circuit Breaker як бібліотека відмовостійкості описує стратегію запобігання каскадному збою на різних рівнях програми, відстежує методи невдалих викликів відповідних служб. У разі такої відмови Circuit Breaker розімкне коло і перенаправить виклик до резервного методу.

Бібліотека стримуватиме збої до певного порогового значення. Крім того, вона залишає коло відкритим. Це означає, що буде перенаправлено всі подальші виклики на резервний метод, запобігаючи майбутнім збоям і створюючи тимчасовий буфер для пов'язаної служби, щоб відновити стан збою.

Отже, з випуском Circuit Breaker більшість розробників у процесі проєктування систем прийшли до ідеї наперед піклуватись про запобігання поширенню помилок з однієї частини системи на всі інші.

Розглянемо для прикладу систему книгарні. Зазвичай це типовий мікросервіс. Домашня сторінка сайту потребує виведення категорії книги, а також персональних рекомендацій для клієнта на основі його статичної моделі. Якщо ця операція неуспішна, є два можливих сценарії: показати клієнту 500 помилок або показати сторінку з рекомендаціями, отриманими з інших місць, чи взагалі без рекомендацій.

Щоб передбачити цю ситуацію, розробник може описати резервний метод, який здійснюється автоматично, якщо виконуються певні умови (наприклад, зовнішня служба повернула 404/500 або взагалі виявлена помилка тайм-ауту). Такі «заглушки» є прозорими для вас, вашої кодової бази та колег, які також підтримують код. Вони є простими у використанні, а отже, потрібно уважно придивитися до такого шаблону оформлення і спробувати використати його там, де це виправдано.

ОСНОВНА ЧАСТИНА

Забезпечення стабільності застосунків із мікросервісною архітектурою з використанням бібліотек Circuit Breaker

Архітектура мікросервісу має передбачати отримання помилок, проте Circuit Breaker розрахований на менш очікувані помилки, які можуть тривати набагато довше: розрив мережі, відмова сервісу, обладнання тощо. У цих ситуаціях під час повторної спроби надіслати аналогічний запит із великою часткою ймовірності ми дістанемо аналогічну помилку. Наприклад, застосунок взаємодіє з деяким сервісом (рис. 1).

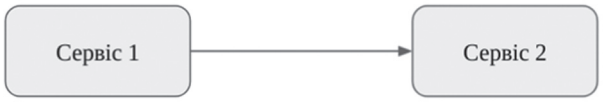

Рис. 1. Сервіси взаємодіють коректно, без збоїв

У межах реалізації запитів та відповідей передбачений деякий тайм-аут, після якого в разі, якщо від сервісу не отримано відповідь, операція вважається неуспішною. Якщо з цим сервісом є проблеми, то під час очікування відповіді та до досягнення тайм-ауту програма може споживати якісь критично важливі ресурси (пам'ять, процесорний час), які ймовірніше потрібні іншим частинам програми (рис. 2).

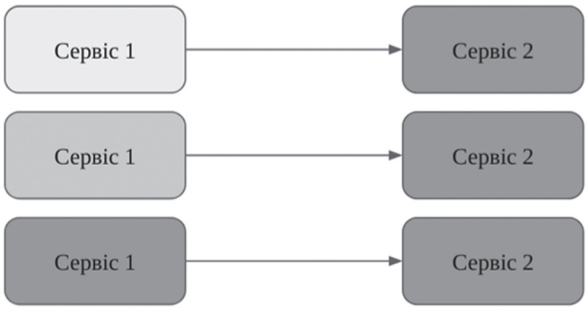

Рис. 2. Подальша взаємодія сервісів після відмови сервісу 2, поступова відмова сервісу 1

У такій ситуації застосунку (сервіс 1) буде краще завершити операцію з помилкою одразу, не чекаючи тайм-ауту від сервісу, і повторювати спробу тільки тоді, коли ймовірність успішного завершення буде досить високою (рис. 3). Для цього використовують певний проксі-сервіс, зокрема Circuit Breaker.

Рис. 3. Взаємодія між сервісами через проксі-сервіс

Circuit Breaker запобігає спробам програми виконати операцію, яка найвірогідніше завершиться невдало, що дає змогу продовжити роботу далі, не витрачаючи важливі ресурси, поки не стане відомо, що проблему не усунено. Програма має швидко прийняти збій операції та обробити його.

Він також дає можливість застосунку визначати, чи було усунено несправність. Якщо проблему вирішено, програма може спробувати викликати операцію знову.

Реалізація Circuit Breaker у мікросервісній архітектурі

Circuit Breaker діє як проксі-сервер операцій, які можуть завершитися зі збоєм. Проксі-сервер має відслідковувати кількість недавніх збоїв і використовувати цю інформацію, щоб вирішити, чи дозволити продовження операції, чи повернути вимкнення.

Проксі-сервер може бути реалізований як перемикач із такими станами:

• ***закритий:*** запит програми перенаправляється на операцію; лічильник збоїв береться таким, що дорівнює 0 та збільшується в разі негативного виконання операції; після перевищення порогу протягом заданого періоду часу проксі-сервер переводиться у стан ***відкритий*** та запускає таймер часу очікування, після закінчення якого переводиться в стан ***напіввідкритий***;

• ***відкритий:*** запит програми завершується зі збоєм, і вимкнення повертається до програми;

• ***напіввідкритий:*** обмеженій кількості запитів від програми можна проходити через операцію та викликати її; якщо ці запити виконуються успішно, то помилку, яка раніше спричинила збій, усунено, а Circuit Breaker перетворюється у стан ***закритий*** (лічильник збоїв скидається); якщо будь-який запит завершується зі збоєм, припускається, що несправність все ще є, тому він повертається до стану ***відкритий*** і перезапускає таймер часу очікування, щоб дати системі додатковий час на відновлення після збою.

Закритий та відкритий стани схематично зображено на рис. 4.

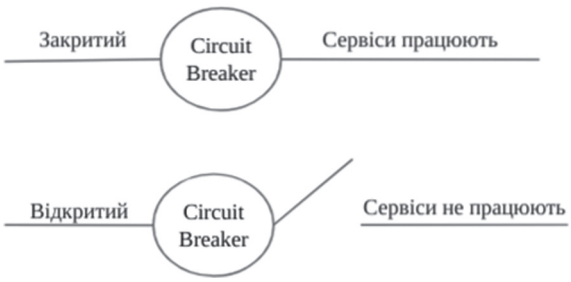

Рис. 4. Закритий та відкритий стани в Circuit Breaker шаблоні

Схему взаємодії всіх трьох станів унаочнює рис. 5.

Рис. 5. Схема станів у Circuit Breaker шаблоні

ВИСНОВКИ

Щоб запобігти спробі виклику програмою віддаленої служби або отриманню доступу до спільного ресурсу, якщо ця операція, найімовірніше, завершиться зі збоєм, рекомендовано використовувати Circuit Breaker. Проте, якщо застосовувати його для оброблення доступу до локальних закритих

ресурсів у програмі, наприклад структурі даних у пам'яті або під час використання автоматичного вимкнення, або коли помилки мають довгостроковий характер, то варто очікувати зростання навантаження в системі, адже програма буде марно витрачати ресурси на спроби повторити операції. Тому для Circuit Breaker варто забезпечити коректне використання лише там, де це потрібно, адже застосунок може забезпечити стабільність роботи вашій мікросервісній системі або ж, навпаки, призвести до збільшення навантаження.

Список використаної літератури

*1. **A Guide** to Spring Cloud Netflix – Hystrix [Електронний ресурс]. URL:*

https://www.baeldung.com/spring-cloud-netflix-hystrix

*2. **Circuit Breaker** pattern [Електронний ресурс]. URL:*

https://docs.microsoft.com/ru-ru/azure/architecture/patterns/circuit-breaker

*3. **Стратегії** обробки помилок: Circuit Breaker pattern [Електронний ресурс]. URL:*

https://medium.com/@kirill.sereda/%D1%81%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D0%B8%D0%B8-%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8-%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA-circuit-breaker-pattern-650232944e37

*4. **Circuit Breaker** паттерн [Електронний ресурс]. URL:*

https://bool.dev/blog/detail/circuit-breaker-pattern

*5. **Техніки** обробки відмов сервісу в мікросервісних архітектурах або Альтернативи Circuit Breaker [Електронний ресурс]. URL:*

https://habr.com/ru/company/arcadia/blog/571442/

*6. **Tushych A. M.** Analysis of problems of production modernization through implementation of IT technologies // Economic trends: new opportunities and threats: International scientific conference. Le Mans, France: Le Mans University, 2021. November 19-20, 2021. P. 58–61.*

*7. **Tushych A.** Analysis of the main problems in the field IT today // The IV International Scientific and Practical Conference «Actual problems of practice and science and methods of their solution». Milan, Italy, 2022. January 31 – February 02, 2022. P. 581–582.*

A. Tushych

USING CIRCUIT BREAKER IN SOFTWARE WITH MICROSERVICE ARCHITECTURE

Microservices is an architectural style in which one program is created as a set of small programs, each of which works in its own process and interacts with others using simple and fast data transfer protocols. In addition to many advantages, this method of designing an architecture has an important disadvantage: if the system runs many separate services, any software node can fail and cause a cascade of errors. To prevent the spread of cascading termination of microservices, Circuit Breaker is used — a library of fault tolerance, which describes the strategy of preventing cascading failure at different levels of the program and monitors the methods of unsuccessful calls to relevant services. The library will keep failures down to a certain threshold. In addition, it leaves the chain open. This means that all subsequent calls will be redirected to the backup method to prevent future failures. This creates a temporary buffer for the associated service to restore the fault state. The architecture of the microservice should provide for errors, but Circuit Breaker is designed for less expected errors, which can last much longer: network failure, service failure, hardware, and so on. In these situations, when we try to send a similar request again, we will most likely get a similar error. The paper describes the schemes of using Circuit Breaker and recommendations for correct operation.

Keywords: microservice architecture; proxy service; Circuit Breaker; switch; fault tolerance.