УДК 004.4:004.75
DOI 10.47049/2226-1893-2022-1-81-89

РЕАЛІЗАЦІЯ МІЖПРОЦЕСНОЇ ВЗАЄМОДІЇ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

І.Г. Бугаєва
к.т.н, доцент кафедри «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Меркта»
М.В. Розум
к.ф.-м.н., доцент кафедри «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Меркта»

Одеський національний морський університет

Анотація. *Однією з основних проблем, які необхідно вирішити під час проектування веб-застосування з мікросервісною архітектурою, є вибір механізму, за допомогою якого сервіси будуть взаємодіяти один з одним. У статті досліджуються синхронний та асинхронний підходи для реалізації міжпроцесної взаємодії, проведено порівняльний аналіз на основі існуючих досліджень для виявлення переваг та недоліків кожного з цих підходів. Розглянуто приклад використання у веб-застосуванні брокера повідомлень RabbitMQ для організації взаємодії між мікросервісами.*

Ключові слова: *мікросервіси, міжпроцесна взаємодія, IPC, міжсервісна взаємодія, REST, RabbitMQ.*

UDC 004.4:004.75
DOI 10.47049/2226-1893-2022-1-81-89

IMPLEMENTING INTERPROCESS COMMUNICATION IN MICROSERVICE ARCHITECTURE

I.G. Buhaieva
Ph.D., Docent of the Department «Technical Cybernetics and Information Technologies named after prof. R.V. Merkt»
M.V. Rozum
Ph.D., Docent of the Department «Technical Cybernetics and Information Technologies named after prof. R.V. Merkt»

Odessa National Maritime University

***Abstract**. One of the main problems that needs to be solved when designing a web application with a microservice architecture is the choice of a mechanism by which services will interact with each other. The article explores synchronous and asynchronous approaches to the implementation of interprocess communication, a comparative analysis is carried out based on existing research to identify the advantages and disadvantages of each of these approaches. An analysis of the results of the experiments showed that asynchronous interaction of microservices implemented on the basis of the AMQP protocol provides better performance and higher availability of microservices. A synchronous form of microservice communication implemented using REST or gRPC can offer higher throughput than an asynchronous method when the system load is relatively light. As the number of service users increases, an asynchronous communication method using message queues is more appropriate. Besides this way of messaging between microservices is more reliable. Message queues are also the best choice when you need to synchronize data between services. The implementation of an asynchronous way of interaction between microservices in a web application using the RabbitMQ message broker is considered.*

***Keywords**: microservices, interprocess communication, IPC, interservice communication, REST, RabbitMQ.*

Вступ. У даний час у зв'язку з розвитком та розповсюдженням мережевих хмарних сервісів розробка веб-застосувань, що базуються на мікросервісній архітектурі, стає все більш популярною. Мікросервісна архітектура – це стиль проектування програмного забезпечення, який розбиває всю систему на дрібніші, неза-лежні та функціональні одиниці – сервіси, кожен з яких може працювати та масштабуватися незалежно.

Щоб обробляти запити, мікросервісам часто доводиться взаємодіяти між собою. Екземпляри сервісів зазвичай є процесами, що запущені на різних комп'ютерах, і для спілкування один з одним вони повинні використовувати механізми, які забезпечують міжпроцесну взаємодію (IPC). Вибір механізму IPC є важливим архітектурним рішенням і може вплинути на рівень доступності застосування.

У мікросервісній архітектурі розрізняють синхронну та асинхронну взаємодію між клієнтом та сервісом: при синхронній – запити клієнтів вимагають негайної відповіді від сервісу і можуть бути заблоковані через очікування, при асинхронній – клієнт не блокується, а відповідь може бути надіслана не відразу. Для реалізації міжпроцесної взаємодії у мікросервісах існує безліч різних технологій. Сервіси можуть використовувати комунікаційні механізми на основі запитів/відповідей, такі, як REST або gRPC, або асинхронні механізми комунікації на основі повідомлень, такі як, наприклад, протокол передачі AMQP. В якості системи обміну повідомленнями, що реалізує й доповнює протокол AMQP, можна навести брокер повідомлень RabbitMQ.

Метою статті є дослідження механізмів комунікації мікросервісів, виявлення переваг та недоліків синхронного та асинхронного підходів для здійснення міжпроцесної взаємодії та реалізація одного з них для організації взаємодії мікросервісів у веб-застосуванні.

Аналіз досліджень і публікацій. Різними авторами було проведено ряд досліджень синхронного та асинхронного способів взаємодії мікросервісів, щоб порівняти та оцінити продуктивність сервісів та їх доступність.

В експериментах, результати яких представлені в статті [1], використовувалися механізми комунікації на основі REST, gRPC та RabbitMQ. Було виконано кілька сценаріїв навантажувального тестування для отримання кількісних даних, що є показниками продуктивності та доступності мікросервісу з використанням синхронних та асинхронних методів взаємодії.

У першій серії тестів вимірювалася пропускна здатність за загальною кількістю запитів та відповідей.Результати тестів показали, що синхронна форма спілкування мікросервісів може запропонувати більш високу пропускну здатність, ніж асинхронний метод, коли навантаження на систему відносно невисоке. Коли кількість віртуальних користувачів збільшилася вчетверо в порівнянні з першим випадком, асинхронний метод зв'язку з використанням RabbitMQ перевершив два інші методи, маючи можливість обробляти 4480 запитів за 180 секунд, тоді як gRPC та RESTAPI вдалося відповідно обробити на 132 та 146 запитів менше.

Доступність сервісу обчислювалася за такою формулою:

$$Availability = \frac{MTTF}{MTTF + MTTR},$$

де $MTTF$ – середній час роботи системи до виникнення збою;

$MTTR$ – середній час відновлення.

На основі цієї формули були виконані три інші тести, які не мали певної тривалості. Вони працювали доти, доки служби ставали недоступними у зв'язку з великою кількістю запитів, що надходили. Перший, другий та третій тести проходили відповідно з 200, 300 та 400 віртуальними користувачами. Результати тестів показали, що мікросервіс на основі асинхронної взаємодії пропонує більшу доступність, ніж його синхронні аналоги.

Оцінка експерименту показала, що в середньому асинхронний підхід забезпечує кращу продуктивність та більш високу доступність мікросервісів.

У статті [2] представлено порівняльне дослідження продуктивності веб-служб REST та протоколу AMQP з урахуванням обміну повідомленнями між клієнтом та сервером. Дослідження ґрунтувалося на середній кількості повідомлень, якими вони обмінювалися за певний період часу. Було зроблено висновок, що при обміні великими обсягами повідомлень найкращі результати дає протокол AMQP.

У статті [3] описано використання RabbitMQ та REST API в якості проміжного програмного забезпечення, що орієнтоване на передачу повідомлень у веб-застосуваннях на основі мікросервісів. Автори проводили експерименти з обома методами з різною кількістю користувачів, щоб порівняти та оцінити їх ефективність у різних обставинах. Отримані експериментальні результати показали, що, коли велика кількість користувачів одночасно надсилають запити до веб-застосування, більш стабільну роботу програми забезпечує використання RabbitMQ для передачі повідомлень, ніж метод зв'язку REST API.

У роботі [4] описано експеримент, у якому порівнювалася продуктивність двох веб-застосувань на основі мікросервісів. Дослідники провели експеримент з різною кількістю користувачів від 50 до 350. Користувачі відправляли запити на інформацію протягом 15 хвилин. Проводилося порівняння швидкості відгуку мікросервісів з використанням REST API та RabbitMQ. Результати тестів показали, що, якщо кількість користувачів дорівнює 50, різниця в швидкості відгуку двох мікросервісів практично незначна. На початкових етапах зі збільшенням кількості користувачів збільшується швидкість відповіді REST API. Але, якщо кількість користувачів збільшити з 250 до 300, швидкість відгуку RabbitMQ значно збільшується. Для 300 користувачів REST API має дуже низьку продуктивність у порівнянні з RabbitMQ.

Автори статті роблять висновок, що сервіс REST може використовуватися як канал зв'язку між мікросервісами лише у разі невеликого трафіку в каналах. При збільшенні трафіку краще використовувати брокер повідомлень RabbitMQ. Навіть при невеликому обсязі трафіку в каналі зв'язку можна запропонувати RabbitMQ, оскільки при взаємодії REST-сервісів можлива втрата частини пакетів. В брокері повідомлень, навіть коли з'єднання між сервісами перервано, дані черги повідомлень зберігаються і можуть бути передані за призначенням, коли з'єднання відновиться.

Виклад основного матеріалу. Аналіз літературних джерел дозволив визначити переваги та недоліки асинхронного та синхронного способів взаємодії мікросервісів.

Розглянемо спочатку реалізацію першого підходу на прикладі використання брокеру повідомлень RabbitMQ, який працює на базі протоколуAMQP.

Протокол AMQP вводить три поняття:

• exchange (обмінник) – повідомлення надсилаються до цієї точки обміну. Обмінник розподіляє повідомлення на одну або кілька черг. Він маршрутизує повідомлення до черги на основі створених зв'язків (binding) між ним та чергою;

• queue (черга) – структура даних на диску або в оперативній пам'яті, яка зберігає посилання на повідомлення та віддає копії повідомлень споживачам (consumers). Одна черга може використовуватися кількома споживачами;

• binding (прив'язка) – правило, яке повідомляє точці обміну, до якої черги ці повідомлення повинні потрапляти.

Основна ідея моделі обміну повідомленнями у RabbitMQ полягає в тому, що видавець (producer) може надсилати повідомлення лише на обмін. З одного боку, обмін отримує повідомлення від видавців, з другого – відправляє їх у черги. Черга зберігає і віддає споживачам всі повідомлення, що надходять. Обмінник займається маршрутизацією повідомлень на основі створених зв'язків між ним та чергами. Для такого обміну інформацією між клієнтом та сервером використовуються канали. Канали створюються у межах певного підключення.

Є чотири типи обмінників:

- directexchange – використовується, коли потрібно доставити повідомлення у певні черги. Повідомлення публікується в обмінник з певним ключем маршрутизації та потрапляє у всі черги, пов'язані з цим обмінником аналогічним ключем маршрутизації;
- topicexchange – дає можливість здійснення вибіркової маршрутизації шляхом порівняння ключа маршрутизації, але в данному випадку ключ задається за шаблоном;
- fanoutexchange – всі повідомлення доставляються в усі черги, навіть якщо в повідомленні заданий ключ маршрутизації;
- headersexchange – спрямовує повідомлення у черги на основі порівняння пар (ключ, значення) властивості headers прив'язки та аналогічної властивості повідомлення.

У RabbitMQ також існує поняття «обмінник» за замовчуванням. Це обмінник типу directexchange без імені. Якщо застосовується стандартний обмінник, то повідомлення будуть маршрутизуватися в чергу з ім'ям, рівним ключу маршрутизації повідомлення.

Використання механізму повідомлень дає такі важливі переваги, як [5]:

- слабка пов'язаність: для виконання запиту клієнта не потрібно використовувати механізм виявлення сервісу, щоб визначити його місцезнаходження;
- буферизація повідомлень: брокер буферизує повідомлення доти, доки їх не зможуть обробити.

Проте механізм повідомлень має свої недоліки:

- додаткова складність експлуатації: систему обміну повідомленнями необхідно встановлювати, налаштовувати та розгортати окремо;
- брокер повідомлень може стати вузьким місцем продуктивності, але багато сучасних брокерів спроектовані з підтримкою високої масштабованості.

При синхронній взаємодії на основі запиту/відповіді клієнт надсилає запит на сервер, а сервер обробляє запит та повертає відповідь. Деякі клієнти будуть заблоковані в очікуванні відповіді від сервера, тоді як інші можуть використовувати асинхронний, керований подіями клієнтський код. Проте, на відміну від використання механізму повідомлень, сервіс повинен відповідати своєчасно.

Один з найпоширеніших механізмів IPC на основі запитів/відповідей-REST. Архітектурний стиль REST [6] використовує протокол HTTP для управління ресурсами та реалізується через URL-адресу. Наприклад, запит GET поверне інформацію про ресурс, який може бути у форматі документа XML або об'єкта JSON. Запит POST створить новий ресурс, PUT відновить ресурс, а DELETE видалить ресурс.

Інший механізм міжпроцесної взаємодії на основі запитів/відповідей – gRPC. Це система віддаленого виклику процедур з відкритим вихідним кодом, що розроблена для забезпечення високошвидкісного зв'язку між мікросервісами. gRPC дозволяє розробникам інтегрувати сервіси, що написані різними мовами, використовує протокол ProtocolBuffers, який є двійковим форматом обміну повідомленнями з високим ступенем упаковки для серіалізації структурованих даних. Protocol Buffersy gRPC працює поверх протоколу HTTP/2.

Використання протоколів на основі HTTP має такі переваги:

- HTTP дуже простий;
- можна використовувати HTTP-клієнт Postmana бо командний рядок, наприклад curl, для тестування API;
- вбудована підтримка зв'язку в режимі запит/відповідь;
- не потрібен проміжний агент, що спрощує архітектуру системи.

До недоліків можна віднести:

- підтримується лише взаємодія в режиміз апиту/відповіді;
- клієнт та сервер повинні залишатися в мережі під час взаємодії;
- клієнт повинен використовувати механізм виявлення екземпляра служби.

Аналіз результатів досліджень показав, що асинхронна взаємодія з іншими сервісами у межах обробки запитів забезпечує кращу продуктивність та більш високу доступність мікросервісів. У зв'язку з цим при проектуванні сервісів краще використовувати асинхронний обмін повідомленнями.

Щоб повністю уникнути проблеми синхронної взаємодії з іншими сервісами, всі сервіси можна забезпечити виключно асинхронними API. Але це не завжди можливо. Наприклад, публічні API зазвичай дотримуються стандарту REST. Тому деякі послуги повинні мати синхронні API.

Якщо сервіс має синхронний API, доступність можна покращити за рахунок того, що він зберігає копію даних, які потрібні для обробки запитів, і йому не потрібно звертатись за ними до інших сервісів. При оновленні даних в основній базі даних відбувається їх оновлення і в базі даних мікросервісу. Черги повідомлень є ідеальним рішенням для синхронізації даних між сервісами.

Реалізацію міжпроцесної взаємодії розглянемо на прикладі двох мікросервісів, що входять до складу веб-застосування для продажу товарів через Інтернет.

Перший мікросервіс призначено для виконання функцій адміністратора: створення ресурсу, яким є товар, його редагування, видалення, видача списку всіх товарів та інформації про товар за його ідентифікатором. Другий мікросервіс видає по запиту користувача інформацію про всі товари.

Веб-застосування на основі мікросервісної архітектури розроблено в середовищі Node.js. Клієнтську частину побудовано за допомогою Javascript-бібліотеки React.js. Зовнішні API-інтерфейси, що надають користувачам застосування обидва сервіси, реалізовано в стилі REST. У кожного мікросервісу своя база даних MySQL, є необхідність синхронізації їх даних. Для організації взаємодії цих мікросервісів використовується брокер повідомлень RabbitMQ.

Основні події, що відбуваються в процесі такої взаємодії:

- після встановлення зв'язку з брокером та створення каналу channel споживач (другий сервіс) створює три черги «product_created», «product_updated» та «product_deleted» за допомогою команди AMQPchannel.assertQueue, підписується на отримання повідомлень, використовуючи команду channel.consume;
- видавець (перший сервіс) надсилає повідомлення безіменному обміннику типу direct;
- обмінник, отримавши повідомлення, маршрутизує його в чергу з ім'ям, рівним ключу маршрутизації повідомлення;
- черга зберігає повідомлення;
- як тільки споживач готовий отримати повідомлення з черги, сервер створює копію повідомлення та надсилає;
- споживач отримує повідомлення та надсилає брокеру підтвердження;
- брокер, отримавши підтвердження, видаляє повідомлення з черги.

У повідомленнях, що передаються при взаємодії мікросервісів, міститься інформація про створений або відредагований товар, або ідентифікатор видаленого товару. Після отримання цих повідомлень другий мікросервіс оновлює свою базу даних відповідно до операції, що було виконано. Користувач при звертанні до другого мікро сервісу отримує актуальний на момент запиту список товарів. При цьому другому мікросервісу не потрібно звертатися за даними до інших служб.

Таким чином, використання брокеру повідомлень забезпечує синхронізацію баз даних мікросервісів. Крім того, забезпечується надійність роботи застосування. Черги повідомлень надають буфер для тимчасового зберігання повідомлень та кінцеві точки, які дозволяють підключатися до черги для надсилання та отримання повідомлень в асинхронному режимі.

Висновки. Досліджено можливі механізми реалізації синхронної та асинхронної міжпроцесної взаємодії у веб-застосуваннях з мікросервісною архітектурою, проведено порівняльний аналіз на основі літературних джерел та існуючих досліджень з цього питання для виявлення переваг та недоліків кожного з цих підходів. Аналіз результатів експериментів показав, що асинхронна взаємодія мікросервісів, що реалізованана основі протоколу AMQP, забезпечує кращу продуктивність та більш високу доступність мікросервісів. Синхронна форма спілкування мікросервісів, яка реалізується за допомогою REST або gRPC, може запропонувати вищу

пропускну здатність, ніж асинхронний метод, коли навантаження на систему відносно невисоке. Коли кількість користувачів сервісу збільшується, асинхронний метод зв'язку з використанням черг повідомлень є більш придатний. Крім того, цей спосіб обміну повідомленнями між мікросервісами є більш надійним. Черги повідомлень є також кращим вибором при необхідності синхронізації даних між сервісами. Розглянуто реалізацію асинхронного способу взаємодії двох мікросервісів у веб-застосуванні за допомогою брокеру повідомлень RabbitMQ.

ЛІТЕРАТУРА

1. *Benyamin Shafabakhsha, Robert Lagerströmb and Simon Hacksb. Evaluating the Impact of Inter Process Communication in Microservice Architectures. Conference: QuASoQ 2020 8th International Workshop on Quantitative Approaches to Software Quality, December 2020.*
2. *Fernandes J.L., Lopes I.C., Rodrigues J.J. and Ullah S. Performance evaluation of RESTful web services and AMQP protocol. Fifth International Conference on Ubiquitous and Future Networks, Da Nang, Vietnam, July 2013.*
3. *Hong X.J., Yang H.S. and Kim Y.H. Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application. International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), October 2018.*
4. *Gurudat K.S. and Padmashree T. A Better Solution TowardsMicroservices Communication In Web Application: A Survey. International Journal of Innovative Research in Computer Science & Technology (IJIRCST), vol. 7, issue: 3, May 2019, pp. 71-74.*
5. *Richardson, Ch. Microservices Patterns: With examples in Java. – Manning Publications, 2019. – 520 p.*
6. *Fielding R. Architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California, Irvine, 2000.*

REFERENCES

1. *Shafabakhsha, B., Lagerströmb, R.,Hacksb, S.Evaluating the Impact of Inter Process Communication in Microservice Architectures. Conference: QuASoQ 2020 8th International Workshop on Quantitative Approaches to Software Quality, December 2020.*
2. *Fernandes,J.L., Lopes,I.C., Rodrigues, J.J.,Ullah,S. Performance evaluation of RESTful web services and AMQP protocol. Fifth International Conference on Ubiquitous and Future Networks, Da Nang, Vietnam, July 2013.*

3. *Hong,X.J., Yang, H.S.,Kim, Y.H. Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application. International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), October 2018.*
4. *Gurudat, K.S. andPadmashree, T. A Better Solution TowardsMicroservices Communication In Web Application: A Survey. International Journal of Innovative Research in Computer Science & Technology (IJIRCST), vol. 7, issue: 3, May 2019, pp. 71-74.*
5. *Richardson, Ch. Microservices Patterns: With examples in Java. –Manning Publications, 2019. – 520p.*
6. *Fielding, R. Architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California, Irvine, 2000.*

Стаття надійшла до редакції 31.08.2022

Посилання на статтю: Бугаєва І.Г., Розум М.В. Реалізація міжпроцесної взаємодії в мікросервісній архітектурі // Вісник Одеського національного морського університету: Зб. наук. праць, 2022. № 1(67). С. 81-89. DOI 10.47049/ 2226-1893-2022-1-81-89.

Article received 31.08.2022

Reference a JournalArtic: Buhaieva I.G., Rozum M.V. Implementing interprocess communication in microservice architecture // Herald of the Odessa national maritime university. 2022. № 1(67). 81-89. DOI 10.47049/ 2226-1893-2022-1-81-89.