УДК 004

Шаблон микросервиса-агрегатора в Java

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается, как шаблон микросервиса-агрегатора можно реализовать на Java с использованием различных подходов, таких как асинхронная связь, синхронная связь или их комбинация. Также будут предоставлены примеры кода для иллюстрации каждого подхода.
Ключевые слова: Java, Шаблон, Микросервис

Aggregator microservice pattern in Java

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Erovleva Regina Victorovna
Sholom-Aleichem Priamursky State University
Student

Abstract
This article explores how the aggregator microservice pattern can be implemented in Java using various approaches such as asynchronous communication, synchronous communication, or a combination of both. Code examples will also be provided to illustrate each approach.
Keywords: Java, Template, Microservice

1 Введение

1.1 Актуальность

Шаблон микрослужбы-агрегатора — это шаблон проектирования, используемый для создания сложной службы путем объединения ответов нескольких независимых микрослужб. Этот шаблон подходит, когда для клиентского запроса требуются данные или функции, распределенные по нескольким микрослужбам. Это может повысить производительность и масштабируемость системы, позволяя каждому микросервису

сосредоточиться на конкретной задаче и снижая рабочую нагрузку на один микросервис.

1.2 Обзор исследований

В.Л. Волушкова провела обзор технологий на примере языка java и привела примеры с подробным описанием для использования фреймворка SpringBoot [1]. М.А. Потовиченко, М.В. Привалов и С.В. Корнев рассмотрели разработку программного продукта, который обеспечивает учет данных посещения занятий студентов, а также защиту их работ [2]. А.Д. Нарижный и Н.Е. Губенко провели сравнительный анализ технологий, которые имеют схожую функциональность и предназначены для одинаковых задач на технологии стеков JavaEE и Spring [3]. Д.В. Козырев, Л.А. Володченкова разработали программный интерфейс серверной части для облачного хранилища данных [4]. А.С. Волков и К.А. Волкова произвели краткий обзор элементов трехуровневой архитектуры для современных Web-приложений [5].

1.3 Цель исследования

Цель исследования – используя язык программирования Java, реализовать шаблон микросервиса агрегатора с использованием различных подходов.

2 Материалы и методы

Работа по реализации шаблонов происходит с использованием языка программирования Java и IDE IntelijIdea.

3 Результаты и обсуждения

В Java шаблон микрослужбы агрегатора может быть реализован с использованием различных подходов, таких как асинхронная связь, синхронная связь или их комбинация:

1. Асинхронная связь

Один из способов реализации шаблона микрослужбы агрегатора в Java – использование асинхронной связи между микрослужбами.

При таком подходе клиент отправляет запрос микрослужбе агрегатора, которая затем параллельно отправляет запросы отдельным микрослужбам.

Каждая микрослужба обрабатывает запрос и отправляет ответ обратно в микрослужбу-агрегатор, которая затем объединяет ответы и возвращает результат клиенту.

Преимущество этого подхода заключается в повышении производительности системы за счет одновременной обработки запросов микрослужбами. Однако для этого требуется использование механизма асинхронной связи, такого как очереди сообщений или архитектуры, управляемые событиями, что может усложнить систему.

```java
public class AsyncAggregatorMicroservice<Request> {
    private final ExecutorService executorService;
    private final Microservice1Client microservice1Client;
    private final Microservice2Client microservice2Client;
    private final Microservice3Client microservice3Client;

    public AsyncAggregatorMicroservice(ExecutorService executorService, Microservice1Client microservice1Client, Microservice2Client microservice2Client, Microservice3Client microser
        this.executorService = executorService;
        this.microservice1Client = microservice1Client;
        this.microservice2Client = microservice2Client;
        this.microservice3Client = microservice3Client;
    }

    public CompletableFuture<AggregatedResponse> processRequest(Request request) {
        CompletableFuture<Response1> response1Future = CompletableFuture.supplyAsync(() -> microservice1Client.processRequest(request), executorService);
        CompletableFuture<Response2> response2Future = CompletableFuture.supplyAsync(() -> microservice2Client.processRequest(request), executorService);
        CompletableFuture<Response3> response3Future = CompletableFuture.supplyAsync(() -> microservice3Client.processRequest(request), executorService);
        return CompletableFuture.allOf(response1Future, response2Future, response3Future) .thenApply(v -> new AggregatedResponse(response1Future.join(),
                response2Future.join(), response3Future.join()));
    }
}
```

Рисунок 1 - Микросервис-агрегатор, который использует асинхронную связь

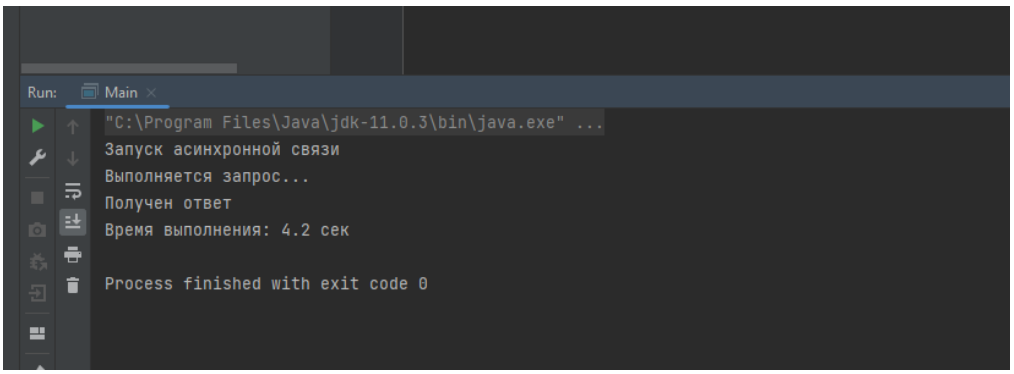

Рисунок 2 – Время выполнения асинхронной связи

В этом примере класс «AsyncAggregatorMicroservice» использует класс «CompletableFuture» из Java «Concurrency API» для асинхронной отправки запросов к отдельным микрослужбам. Метод «CompletableFuture.allOf()» используется для ожидания получения всех ответов, а метод «thenApply()» используется для объединения ответов и возврата результата клиенту.

2. Синхронная связь

Другой способ реализации шаблона микрослужбы агрегатора в Java — использование синхронной связи между микрослужбами. При таком подходе клиент отправляет запрос микрослужбе агрегатора, которая затем последовательно отправляет запросы отдельным микрослужбам.

Каждая микрослужба обрабатывает запрос и отправляет ответ обратно в микрослужбу-агрегатор, которая затем объединяет ответы и возвращает результат клиенту.

Преимуществом этого подхода является простота, поскольку он не требует использования механизмов асинхронной связи. Однако это может негативно сказаться на производительности системы, так как микросервис-агрегатор должен ждать, пока каждый микросервис выполнит свою задачу, прежде чем переходить к следующему.

```java
public class SyncAggregatorMicroservice {
    private final Microservice1Client microservice1Client;
    private final Microservice2Client microservice2Client;
    private final Microservice3Client microservice3Client;

    public SyncAggregatorMicroservice(Microservice1Client microservice1Client, Microservice2Client microservice2Client, Microservice3Client microservice3Client) {
        this.microservice1Client = microservice1Client;
        this.microservice2Client = microservice2Client;
        this.microservice3Client = microservice3Client;

    }

    public AggregatedResponse processRequest(Request request) {
        Response1 response1 = microservice1Client.processRequest(request);
        Response2 response2 = microservice2Client.processRequest(request);
        Response3 response3 = microservice3Client.processRequest(request);
        return new AggregatedResponse(response1, response2, response3);

    }
}
```

Рисунок 3 - Микросервис-агрегатор, который использует синхронную связь

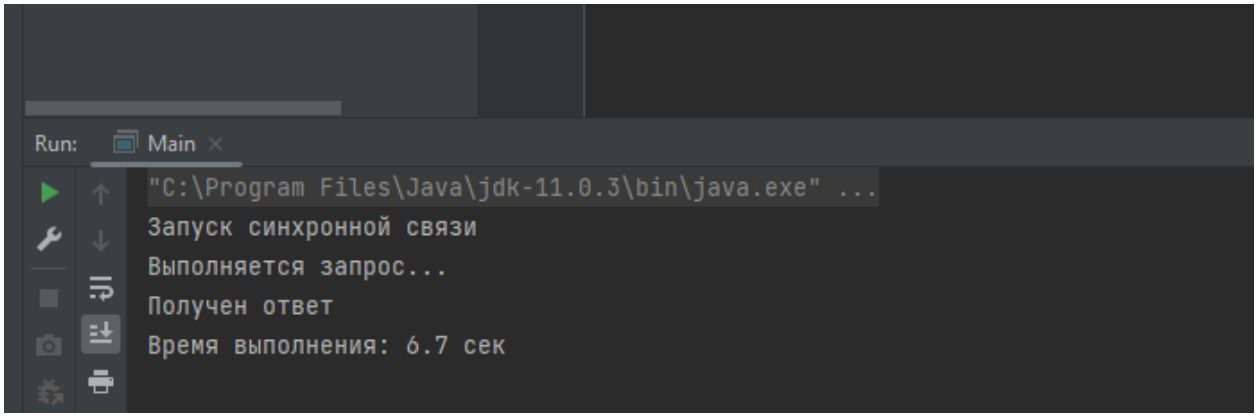

Рисунок 4 – Время выполнения синхронной связи

В этом примере класс «SyncAggregatorMicroservice» отправляет запросы к отдельным микрослужбам синхронно, один за другим. Затем ответы агрегируются и возвращаются клиенту.

3. Сочетание асинхронной и синхронной связи

Также можно реализовать шаблон микросервиса агрегатора на Java, комбинируя асинхронную и синхронную связь. При таком подходе клиент отправляет запрос микросервису-агрегатору, который затем отправляет запросы одним микросервисам асинхронно, а другим синхронно, в зависимости от требований системы.

Этот подход обеспечивает баланс между производительностью и простотой, поскольку он позволяет микрослужбам обрабатывать запросы одновременно, где это возможно, сохраняя при этом простоту реализации.

```java
public class HybridAggregatorMicroservice {
    private final ExecutorService executorService;
    private final Microservice1Client microservice1Client;
    private final Microservice2Client microservice2Client;
    private final Microservice3Client microservice3Client;

    public HybridAggregatorMicroservice(ExecutorService executorService,
                                        Microservice1Client microservice1Client,
                                        Microservice2Client microservice2Client,
                                        Microservice3Client microservice3Client) {

        this.executorService = executorService;
        this.microservice1Client = microservice1Client;
        this.microservice2Client = microservice2Client;
        this.microservice3Client = microservice3Client;
    }

    public AggregatedResponse processRequest(Request request) {
        CompletableFuture<Response1> response1Future = CompletableFuture.supplyAsync(() -> microservice1Client.processRequest(request), executorService);
        Response2 response2 = microservice2Client.processRequest(request);
        CompletableFuture<Response3> response3Future = CompletableFuture.supplyAsync(() -> microservice3Client.processRequest(request), executorService);
        return CompletableFuture.allOf(response1Future, response3Future) .thenApply(v -> new AggregatedResponse(response1Future.join(), response2, response3Future.join()));

    }

}
```

Рисунок 5 – Микросервис-агрегатор, который использует комбинацию асинхронной и синхронной связи

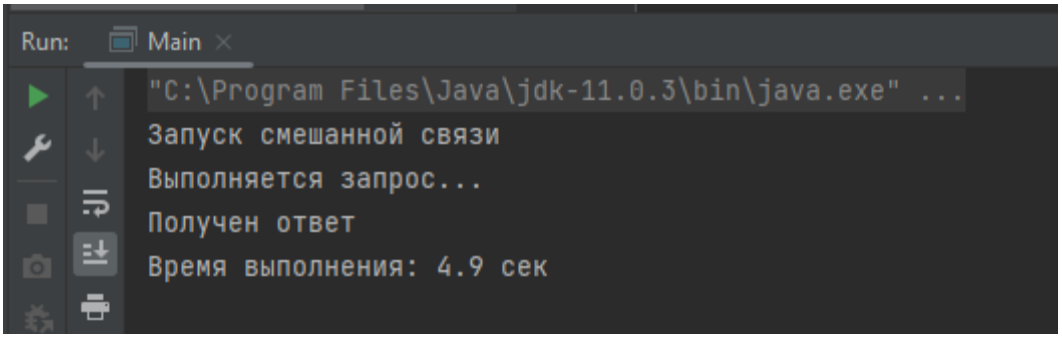

Рисунок 6 – Время выполнения смешанной связи

В этом примере класс «HybridAggregatorMicroservice» отправляет запросы к «microservice1Client» и «microservice3Client» асинхронно, а к «microservice2Client» — синхронно. Затем ответы агрегируются и возвращаются клиенту.

Выводы

Шаблон микросервиса-агрегатора — это полезный шаблон проектирования для составления сложных сервисов путем агрегирования ответов нескольких независимых микросервисов. В Java этот шаблон может быть реализован с использованием асинхронной связи, синхронной связи или их комбинации, в зависимости от требований системы.

Асинхронная связь может повысить производительность системы, но требует использования дополнительных механизмов связи. Синхронную связь проще реализовать, но она может отрицательно сказаться на производительности.

Сочетание асинхронной и синхронной связи обеспечивает баланс между производительностью и простотой.

Библиографический список

1. Волушкова В.Л. Архитектурные решения java для доступа к данным // Тверской государственный университет. 2019. С. 137-140.

2. Потовиченко М.А., Привалов М.В., Корнев С.В. Компьютеризированная подсистема учета текущей успеваемости студента в условиях вуза // Информатика, управляющие системы, математическое и компьютерное моделирование. 2019. № 2. С. 71-75.
3. Нарижный А.Д., Губенко Н.Е. Сравнительный анализ стеков технологий spring и javaee (jakartaee) для разработки enterprise приложений // Информатика, управляющие системы, математическое и компьютерное моделирование. 2020. № 17. С. 459-462.
4. Володченкова Л.А., Козырев Д.В. Разработка серверной части программного приложения для удаленного хранения данных// Математические структуры и моделирование. 2020. № 1 (53). С. 108-138.
5. Волков А.С., Волкова К.А. Обзор архитектурных компонентов современного веб-приложения // Аллея науки. 2019. № 1(28). С. 958-961.