ПОДХОД CQRS В WEB-ПРИЛОЖЕНИЯХ НА МОНОЛИТНОЙ АРХИТЕКТУРЕ

Рынкевич А.А.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Серебряная Л.В. – канд. тех. наук, доцент

Работа посвящена архитектурному приему CQRS, применение которого позволяет создавать гибкие и производительные программные системы. Рассмотрено использование CQRS в web-приложениях на монолитной архитектуре. Показано положительное влияние подхода CQRS на модели программных приложений, а также его удачное сочетание с приемами программирования, улучшающими характеристики различных информационных систем.

Термин «CQRS» «Command and Query Responsibility Segregation» (CQRS) был впервые введен Грегом Янгом и определен им как прием программирования, при котором для совершения операций чтения и записи используются два разных объекта: один – для выполнения команд, другой – для выполнения запросов [1]. Несмотря на всю простоту данного приема, применение CQRS в сочетании с некоторыми другими подходами позволяет сделать систему более производительной и гибкой.

В большинстве архитектур веб-приложений для выполнения операций чтения, записи, изменения или удаления используется одна модель данных. В то время, как это решение отлично применяется в простых системах, в более сложных могут возникнуть следующие проблемы:

1. Формат объектов данных, возвращаемых приложением, сильно отличается от формата, в котором данные хранятся в базе. Время получения таких объектов относительно велико из-за множества операций объединения и большого числа запросов к БД.
2. Требования к скорости выполнения операций чтения и записи отличаются, в то время как модель данных не позволяет удовлетворить их для обеих операций одновременно.
3. Существуют несколько групп пользователей, каждая из которых имеет разные права на получение, добавление, изменение или удаление сущности одного типа. Для создания такого механизма доступа необходима сложная логика, а значит, она больше подвержена ошибкам, что представляет угрозу безопасности хранимой информации.

Хотя CQRS чаще всего используется в системах на микросервисной архитектуре, его можно использовать и в классических монолитных веб-приложениях для решения описанных выше проблем.

Разделим команды и запросы не только на уровне бизнес-логики, но и на уровне базы данных. Схема получившегося приложения в сравнении с классическим подходом представлена на рисунке 1.

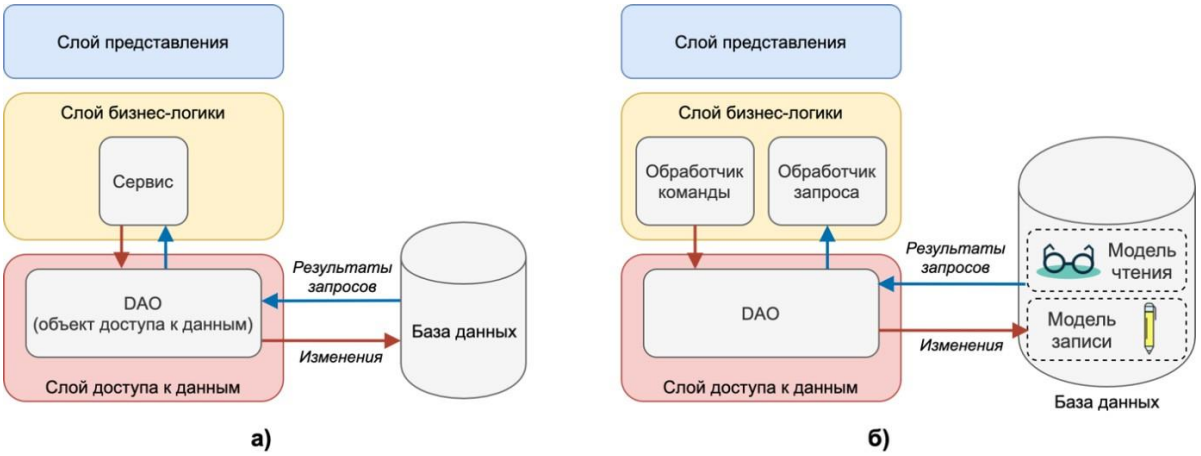

Рисунок 1 – Многослойное веб-приложение без использования CQRS с одной моделью данных (а) и с использованием CQRS и отдельными моделями данных для чтения и записи (б)

Отдельные модели чтения и записи позволяют хранить информацию в том представлении, в

котором это более удобно для каждой из операций. Алгоритм получения информации сводится к отправке простого запроса к хранилищу для получения информации в «готовом» виде.

Стоит отметить, что этот подход предусматривает необходимость синхронизации моделей данных. Однако логика синхронизации может быть вынесена в отдельное фоновое приложение (например, Windows Service) или модуль монолитного веб-приложения с планировщиком задач (Quartz для программ на Java, Quartz.NET для программ на языках платформы .NET или другие аналогичные решения) и выполняться асинхронно.

Асинхронные операции и модель чтения с оптимизированными для представления данными позволяют не только решить проблему снижения производительности из-за отличающихся форматов данных. Они также дают возможность нескольким командам разработчиков параллельно заниматься написанием логики синхронизации и реализацией взаимодействующих с данными модулей.

Развивая идею разделения чтения и записи, можно разместить модели в отдельных базах данных, причем не обязательно одинаковых: например, изменения хранить в реляционной БД, а информацию для чтения – в документной или графовой с подходящими индексами. При выборе способа хранения информации необходимо учитывать характер данных и совершаемые с ними действия, так как от этого зависят средства, которыми будет достигаться уменьшение времени синхронизации и получения информации.

Пример подобного размещения изображен на рисунке 2.

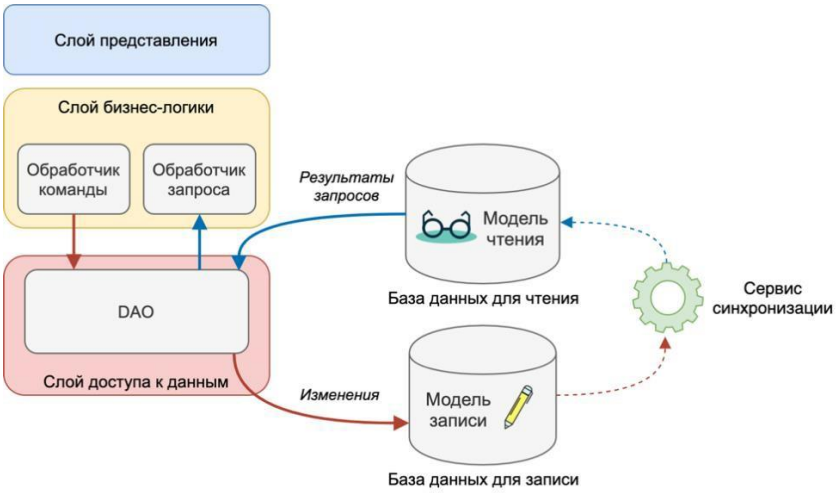

Рисунок 2 – Многослойное веб-приложение с использованием CQRS, отдельными моделями и базами данных для чтения и записи

Адаптируя каждую модель в отдельности, можно удовлетворить требования к скорости выполнения соответствующих операций. Вместе с этим становится проще управление правами и доступом, ведь для каждой сущности выполняется настройка ограничений либо на чтение, либо на запись.

Подход CQRS отлично сочетается с приемом программирования Event Sourcing. Последний предполагает хранение состояния приложения в виде последовательности событий – структур, каждая из которых описывает изменение данных в системе. Для получения актуального состояния системы события «проигрываются» в порядке их поступления. Это позволяет проще обнаруживать и обрабатывать конфликты изменения, а также организовать аудит изменений данных, что может быть важно для администрирования системы. При использовании CQRS модель записи может использоваться как хранилище событий.

Для примера, в банковской информационной системе, использующей Event Sourcing, в качестве событий могут быть представлены транзакции. Тогда суммы на счетах для отображения операционистам и клиентам будут подсчитываться посредством применения событий за сегодняшний день к состоянию счета на вчера, а при закрытии банковского дня, если не наблюдается никаких расхождений в данных, полученные после применения событий состояния счетов будут сохранены в базу.

Использование CQRS можно увидеть во многих архитектурах современных веб-приложений. Одной из них является «Clean Architecture», еще известная как «гексагональная», «луковая» или «порты и адаптеры». Это предметно-ориентированная архитектура, направленная на слабую связность модулей и инверсию зависимостей. В ней CQRS используется для упрощения системы и улучшения ее производительности и масштабируемости.

Таким образом, Command and Query Responsibility Segregation является актуальным архитектурным решением для улучшения характеристик различных информационных систем, в том числе и монолитных веб-приложений.

Список использованных источников:

1. CodeBetter; “CQRS, Task Based UIs, Event Sourcing agh!” by Greg Young [Электронный ресурс] – Режим доступа: http://codebetter.com/gregyoung/2010/02/16/cqrs-task-based-uis-event-sourcing-agh

2. Martin Fowler: CQRS [Электронный ресурс] – Режим доступа: https://martinfowler.com/bliki/CQRS.html

3. CQRS Pattern. Azure Architecture Center | Microsoft Docs [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs

4. E-book “Architect Modern Web Applications with ASP.NET Core and Azure” by Steve “ardalis” Smith [Электронный ресурс] – Режим доступа: https://aka.ms/MicroservicesEbook