ПАТТЕРН АРХИТЕКТУРЫ CQRS В ПРОЕКТИРОВАНИИ ИНФОРМАЦИОННЫХ СИСТЕМ

Бурцев Т.Р., студент**,**

Тазетдинов Б.И., к.ф.-м.н. доцент,

Бирский филиал УУНиТ, г. Бирск, Россия

Аннотация. В данной научной статье рассматривается паттерн архитектуры CQRS в проектировании информационных систем. CQRS разделяет операции записи и чтения данных, что позволяет оптимизировать производительность. В статье рассматриваются основные принципы CQRS, его преимущества и недостатки.

Ключевые слова: паттерн архитектуры CQRS, информационная система, команды и запросы, чтение и запись.

Скорость и качество работы разработчиков могут изменяться в зависимости от их опыта и используемых технологий. В процессе проектирования программного обеспечения отсутствуют строгие стандарты, и разработчики имеют свободу выбора подходов к созданию своих программ. Применение паттерна архитектуры CQRS является одним из наилучших способов увеличить эффективность работы и улучшить производительность программистов.

Архитектурный паттерн CQRS (Command Query Responsibility Segregation) является популярным подходом в проектировании информационных систем, который предлагает разделять операции чтения (query) и записи (command) данных. В этой статье мы рассмотрим, что такое CQRS, его основные принципы и преимущества, а также примеры использования этого паттерна в различных информационных системах.

CQRS - это архитектурный паттерн, который предлагает разделить операции чтения и записи данных в информационной системе. Подход CQRS предполагает обработку операций чтения и операций записи независимо друг от друга, что позволяет использовать отдельные модели данных и логику для каждого типа операций. Это помогает оптимизировать систему под конкретные потребности запросов и операций, а также создать более гибкую и масштабируемую архитектуру.

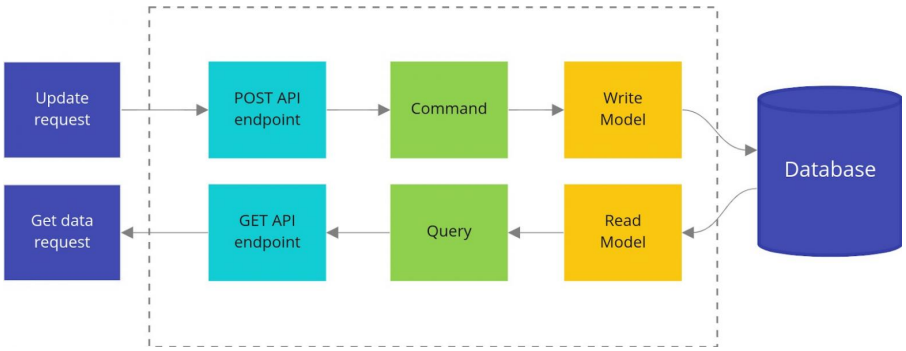

Рисунок 1. Многослойная архитектура CQRS.

Паттерн архитектуры CQRS был описан в 2010 году в статье Грега Янга и быстро стал популярным в разработке приложений, и сегодня является одним из ключевых подходов в разработке сложных систем.

Основные принципы CQRS:

1. Разделение операций чтения и записи: CQRS предлагает явно разделять операции чтения (получения состояния) и операции записи (изменения состояния). Это позволяет оптимизировать систему для эффективного выполнения различных запросов. Команды обычно выполняются с использованием синхронных запросов, что позволяет более точно управлять изменениями состояния и обеспечивает более высокую согласованность данных. С другой стороны, запросы могут быть обработаны с использованием асинхронных запросов для улучшения производительности и масштабируемости приложений.

2. Модели данных: CQRS позволяет использовать разные модели данных для операций чтения и записи. Это означает, что данные, которые предназначены для отображения пользователю, могут быть структурированы и оптимизированы иначе, чем данные, которые изменяются в результате операций записи.

3. Асинхронная обработка команд: В контексте CQRS, операции записи (command) могут обрабатываться асинхронно, что обеспечивает лучшую производительность и масштабируемость системы.

4. Событийно-ориентированная архитектура: CQRS часто используется в сочетании с паттерном Event Sourcing, где каждое изменение состояния данных представляется как событие. Это позволяет легко воссоздавать состояние

системы на любой момент времени и обеспечивает надежную аудиторскую логику.

5. В CQRS отказываются от ORM (Object-Relational Mapping) в пользу использования агрегатов (Aggregates), которые представляют связанные сущности в приложении и содержат логику их изменения состояния. Это позволяет улучшить производительность и масштабируемость, так как ORM может стать узким местом при работе с большим объемом данных.

Компоненты CQRS:

Компоненты CQRS можно разделить на две категории: компоненты записи и компоненты чтения.

Компоненты записи включают в себя:

Command - объект, содержащий информацию для осуществления операции записи в системе и может быть отправлен из различных частей системы, таких как веб-приложение или другой сервис. В C# для создания Command часто используется паттерн Command Object.

Command Handler - компонент, который получает Command и выполняет операцию записи в системе, извлекая данные из Command и передавая их в Write Model для сохранения изменений в базе данных. В C# для обработки Command Handler применяется паттерн Command Handler.

Write Model - модель данных, используемая для операций записи в системе и содержащая необходимые данные для сохранения изменений в базе данных. В C# для создания Write Model часто используется паттерн Repository.

Event - объект, содержащий данные о изменениях в системе после выполнения операции записи и может быть отправлен в другие части системы, заинтересованные в этих изменениях. В C# для создания Event часто используется паттерн Domain Event.

Компоненты чтения включают в себя:

Read Model - это модель данных, предназначенная для чтения, в которой содержатся только необходимые данные для отображения пользователю или выполнения запросов. Она может быть оптимизирована для конкретных запросов, что улучшает производительность системы.

Event Handler - это компонент, который получает события (Event) и обновляет данные в Read Model в соответствии с изменениями в системе. В C# для обработки Event Handler используется паттерн Event Handler.

Query - это объект, содержащий данные, необходимые для выполнения операций чтения в системе. Он может быть отправлен из различных частей системы, таких как веб-приложение или другой сервис. Часто для его создания используется паттерн Query Object.

Query Handler - это компонент, который получает запрос (Query) и возвращает данные из Read Model в соответствии с запросом. Он получает данные из Read Model и возвращает их в виде списка или другого формата, который может быть использован для отображения данных пользователю или выполнения запросов на чтение. В C# для обработки Query Handler используется паттерн Query Handler.

Преимущества CQRS:

1. Гибкость: CQRS позволяет оптимизировать систему для конкретных задач, что повышает гибкость и возможность изменения архитектуры в дальнейшем.

2. Масштабируемость: Благодаря явному разделению операций чтения и записи, система может быть эффективно масштабирована для обработки больших объемов запросов.

3. Производительность: Оптимизация операций чтения и записи позволяет создавать более производительные и отзывчивые системы.

4. Аудит: Использование событийно-ориентированной архитектуры позволяет легко вести аудит изменений в данных и обеспечивает лучшую надежность и отслеживаемость системы.

Недостатки CQRS:

1. Высокая сложность: Реализация CQRS требует более сложной архитектуры и более высокого уровня квалификации у разработчиков, что может усложнить процесс создания и поддержки приложений.

2. Дополнительные расходы: Применение CQRS может потребовать дополнительных затрат на разработку и поддержку, что может сказаться на бюджете проекта.

3. Не универсальное решение: CQRS может быть не подходящим для всех видов приложений, особенно для тех, у которых нет сложной логики команд и запросов, или для проектов с небольшим объемом данных.

В заключение, архитектурный паттерн CQRS представляет собой мощный инструмент для проектирования информационных систем, который позволяет оптимизировать систему для конкретных задач, повышая гибкость, масштабируемость и производительность. Применение CQRS требует тщательного проектирования и анализа потребностей системы, однако при правильном использовании он может значительно улучшить архитектуру и функциональность информационных систем.

Литература

1. Greg Young — CQRS Documents by Greg Young URL: http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf (дата обращения: 23.03.2024)
2. Martin Fowler — CQRS URL: http://martinfowler.com/bliki/CQRS.html (дата обращения: 23.03.2024)
3. Архитектура CQRS. URL: https://habr.com/ru/companies/otus/articles/747668/ (Дата обращения: 25.03.2024)
4. Высоконагруженные приложения. Программирование масштабирование поддержка. Клеппман Мартин, 2018. — 616 р.
5. Паттерн CQRS: теория и практика в рамках ASP.Net Core 5. URL: https://habr.com/ru/articles/543828/ (Дата обращения: 25.03.2024)
6. Чистая архитектура. Искусство разработки программного обеспечения. Роберт Мартин. — Питер, 2018. — 410 с.