# ОСОБЕННОСТИ ПРИМЕНЕНИЯ ПАТТЕРНА CQRS В СОБЫТИЙНО-ОРИЕНТИРОВАННОЙ АРХИТЕКТУРЕ ВЫСОКОНАГРУЖЕННЫХ РАСПРЕДЕЛЕННЫХ СИСТЕМ

PECULIARITIES OF USING THE CQRS PATTERN IN THE EVENT-DRIVEN ARCHITECTURE OF HIGH-LOAD DISTRIBUTED SYSTEMS

**УДК 004.415.2**

**Хитров Николай Олегович,** студент 4 курс, институт «Информационные технологии», МИРЭА - Российский технологический университет, Россия, г. Москва

**Горбачев Максим Андреевич,** студент 4 курс, институт «Информационные технологии», МИРЭА - Российский технологический университет, Россия, г. Москва

**Khitrov N.O.**

**Gorbachev M.A.**

**Аннотация:** В статье рассматриваются паттерны разработки распределенных высоконагруженных систем. Системы подобного рода характеризуются высокими требованиями, где основными критериями являются такие аспекты как масштабируемость, отказоустойчивость и согласованность.

Одним из наиболее распространенных паттернов на сегодняшний день является CQRS (Command Query Responsibility Segregation). Он позволяет оптимизировать операции чтения и записи данных посредством их разделения. Из-за особенностей реализации этот паттерн часто применяется вместе с EDA

(Event-Driven Architecture). В работе рассматривается способ организации архитектуры приложения с применением описанных паттернов.

**Annotation:** The article considers patterns of the development of distributed high-volume systems. Such systems are characterized by high standards with aspects of scalability, fault tolerance, coherence as the main criteria.

The most common pattern today is CQRS (Command Query Responsibility Segregation). It lets streamline reading and data entry operations by separating them. Due to its realization features this pattern is often used with EDA (Event-Driven Architecture).

The article considers patterns of the development of distributed high-volume systems. Such systems are characterized by high standards with aspects of scalability, fault tolerance, coherence as the main criteria.

The most common pattern today is CQRS (Command Query Responsibility Segregation). It lets streamline reading and data entry operations by separating them. Due to its realization features this pattern is often used with EDA (Event-Driven Architecture).

The paper focuses on the way of organizing of application architecture with the use of the described above patterns.

**Ключевые слова:** CQRS, EDA, MVC, очередь сообщений, распределенные системы, масштабируемые системы.

**Keywords:** CQRS, EDA, MVC, message queue, distributed systems, scalable systems.

Шаблон Command-Query Responsibility Segregation (CQRS) по сути своей похож на не менее популярный шаблон MVC (Model View Controller). Он также использует такие понятия как отображение (View), контроллер (Controller). Отличием является то, что CQRS предлагает разделение работы с данными на запросы (Query) и команды (Commands) (см. рисунок 1).

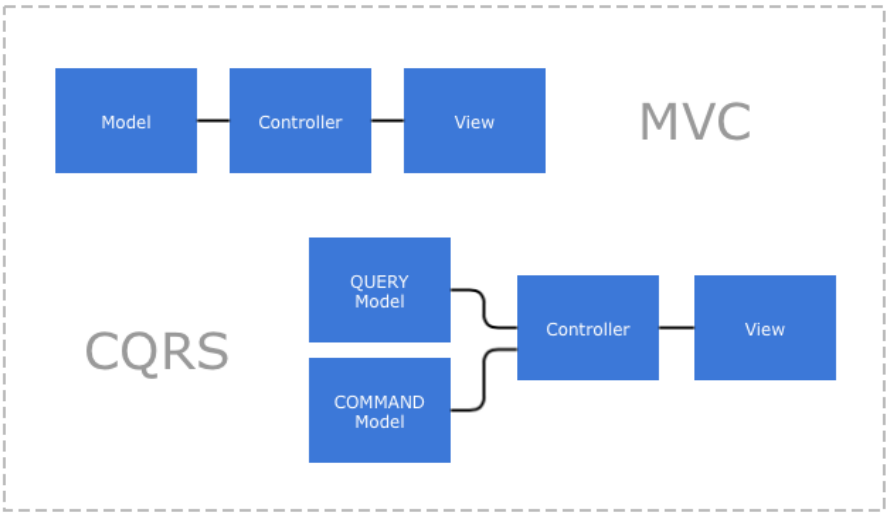


Рисунок 1 – Сравнение MVC и CQRS

Данный подход вводит следующие правила:

1. Запросы используются для получения данных;
2. Запросы не могут изменять состояние;
3. Данные для чтения могут храниться в другой форме и в другом хранилище (но не обязательно);
4. Запросы могут читать данные из хранилища напрямую (отсутствует слой бизнес логики);
5. Команды могут изменять состояние;
6. Команды могут возвращать мета информацию в качестве результата.

Query (Запрос) представляет собой классический поведенческий паттерн “команда”: класс (или модуль, если рассматривать функциональные языки программирования) с одним методом execute(). Запрос позволяет читать необходимую информацию из хранилища данных напрямую, без использования ORM. Это позволяет писать оптимизированные запросы в хранилище для данных, которые необходимо читать максимально быстро.

Команда (Command) представляет собой тот же паттерн. Команда выполняет запрос к системе выполнить определенное действие, например, создать или изменить определенную сущность в системе. Команда, как обработчик запроса, может отказать в исполнении по причине не валидных данных или не валидного состояния запроса.

Общая архитектура приложения. С точки зрения передачи данных, Команду стоит рассматривать в качестве сообщения (message). За обработку сообщения-команды отвечает такая сущность, как Обработчик Команд (Command Handler). Он обрабатывает возможные сценарии выполнения команды и генерирует одно или несколько событий (events).

События являются такими же сообщениями с точки зрения передачи данных, как и Команды. Отличие их состоит в том, что Команды описывают действие, которое необходимо совершить, а события - факт того, что действие произошло.

Также, как и у команд, у событий есть свои обработчики (Event Handlers). Его основная задача состоит в том, чтобы поддерживать данные согласованными (Eventual Consistency) между компонентами системы. На основе полученных событий он может генерировать свои собственные события, предназначенные для других компонентов.

И обработчики команд, и обработчики событий должны работать в фоне, поэтому их необходимо запускать в отдельных потоках или процессах, в зависимости от специфики языка программирования. Например, в языке Elixir это реализовать довольно просто, достаточно воспользоваться модулем GenServer из стандартной библиотеки OTP (Open Telecom Platform). В Python же, если рассмотреть его как альтернативный пример, это сделать сложнее, т.к. язык имеет определенные ограничения на использование потоков из-за наличия GIL (Global Interpreter Lock). Можно воспользоваться модулем multiprocessing, но в таком случае придется столкнуться с некоторыми сложностями при передаче между процессами различных объектов, например клиентских соединений. Можно также воспользоваться модулем asyncio, который позволяет запускать различные задачи в python асинхронно, однако для этого код должен быть написан с использованием синтаксиса async/await. В противном случае снова придется столкнуться с различными проблемами конфигурации и производительности.

Для передачи сообщений обычно используют специальные брокеры, такие как Apache Kafka, RabbitMQ, ZeroMQ и т.д. Они позволяют хранить сообщения в очередях, распределять их по обработчикам. Некоторые из них имеют механизмы повторной отправки и гарантии доставки. Брокеры можно использовать как для передачи сообщений между микросервисами, так и в контексте одного сервиса, например между процессами, что бывает очень удобно.

Для улучшения отказоустойчивости и пропускной способности рекомендуется использовать отдельные соединения для обработчиков команд и событий.

Как правило, операций на чтение сильно больше, чем операций на запись. Поэтому можно добиться сильного увеличения быстродействия системы с использованием кэша. Обновлением кэшей занимается обработчик событий. В качестве базы данных под кэш обычно используют Redis или CouchDB. Некоторые языки программирования предоставляют встроенные im-memory базы данных, например ETS (Erlang Term Storage) для Erlang/Elixir.

Вся схема взаимодействия описанных компонентов представлена на рисунке 2.

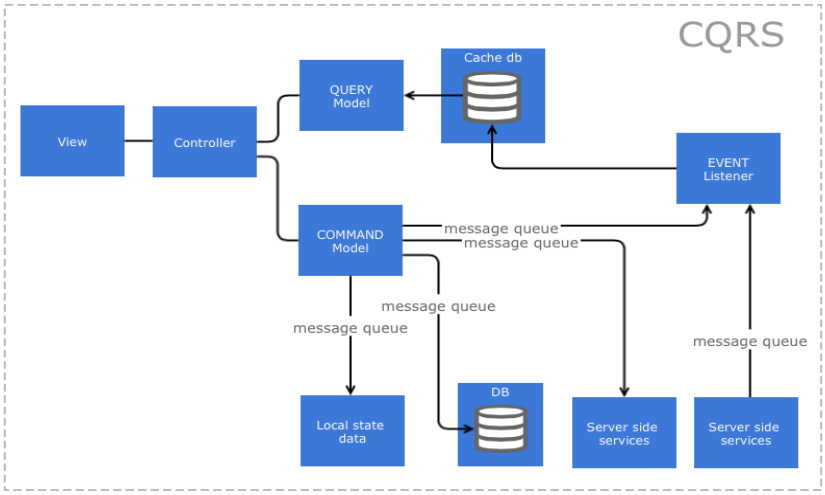


Рисунок 2 – Архитектура приложения с использованием CQRS и событийно-ориентированной модели

## Заключение

В статье был рассмотрен паттерн CQRS с применением событийно-ориентированной архитектуры. Результаты анализа показали, что данный подход способствует упрощению архитектуры компонентов системы благодаря разделению операций записи и чтения. Таким образом уменьшается сцепление и повышается связность модулей, что способствует дальнейшему изменению кода и его оптимизации, что очень важно для высоконагруженных систем. Помимо этого, использование данного подхода упрощает масштабирование системы, поскольку для записи и для чтения можно использовать оптимальный тип базы данных для соответствующей операции.

Таким образом, можно сделать вывод, что паттерн CQRS с применением EDA архитектуры отлично подходит для высоконагруженных систем.

## Литература

1. Greg Young — CQRS Documents by Greg Young [Электронный ресурс] URL: http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf (дата обращения: 26.06.2021).
2. Dominic Betts, Julián Domínguez, Grigori Melnik, Fernando Simonazzi, Mani Subramanian, Foreword by Greg Young — Exploring CQRS and Event Sourcing [Электронный ресурс] URL: http://www.microsoft.com/enus/download/details.aspx?id=34774 (дата обращения: 26.06.2021).
3. Martin Fowler — CQRS [Электронный ресурс] URL: http://martinfowler.com/bliki/CQRS.html (дата обращения: 26.06.2021).
4. Dominic Betts, Julián Domínguez, Grigori Melnik, Fernando Simonazzi, Mani Subramanian, Foreword by Greg Young — Exploring CQRS and Event Sourcing [Электронный ресурс] URL: http://www.microsoft.com/enus/download/details.aspx?id=34774 (дата обращения: 26.06.2021).
5. Высоконагруженные приложения. Программирование масштабирование поддержка. Клеппман Мартин, 2018. — 616 р.

6. Чистая архитектура. Искусство разработки программного обеспечения. Роберт Мартин. — Питер, 2018. — 410 с.

**Literature**

1. Greg Young — CQRS Documents by Greg Young [Electronic resource] URL: http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf (date of the appeal: 26.06.2021)
2. Dominic Betts, Julián Domínguez, Grigori Melnik, Fernando Simonazzi, Mani Subramanian, Foreword by Greg Young — Exploring CQRS and Event Sourcing [Electronic resource] URL: http://www.microsoft.com/enus/download/details.aspx?id=34774
3. Martin Fowler — CQRS [Electronic resource] URL: http://martinfowler.com/bliki/CQRS.htm (date of the appeal: 26.06.2021)l.
4. Dominic Betts, Julián Domínguez, Grigori Melnik, Fernando Simonazzi, Mani Subramanian, Foreword by Greg Young — Exploring CQRS and Event Sourcing [Electronic resource] URL: http://www.microsoft.com/enus/download/details.aspx?id=34774 (date of the appeal: 26.06.2021).
5. Designing Data-Intensive Applications. Martin Kleppmann, 2017. — 616 p.
6. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Robert C. Martin, 2018. — 410 p.