ШАБЛОНЫ РАСПРЕДЕЛЕНИЯ ОТВЕТСТВЕННОСТИ ПО ЗАПРОСАМ КОМАНД И ИСТОЧНИК СОБЫТИЙ

Кива В. С., Борисов Д. В.
Кафедра программного обеспечения информационных технологий, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: knijaz@gmail.com, divlboris@gmail.com

В статье описаны достоинства и недостатки шаблона сегрегация ответственности по запросам команд и шаблона источник событий. Эти шаблоны используются для решения таких проблем как: проблема масштабируемости всего приложения и проблема индивидуальной масштабируемости сторон приложения использующихся как для записи, так и для чтения в системах обработки информации.

Введение

CQRS (сегрегация ответственности запросов и команд) – это очень простой шаблон. Он является наследником шаблона CQS(разделение командных запросов), который был разработан Бертраном Мейером. В соответствии с этим шаблоном методы в системе должны быть разделены на две группы: команды, изменяющие состояние, и запросы, возвращающие значение. Применение этой концепции к объектам или компонентам вводит новую концепцию – CQRS, представленную Грегом Янгом. Основная идея заключается в том, что классы или компоненты внутри приложения, меняющие состояние (команды), должны быть отделены от компонентов, которые получают состояние приложения (запросы).[1]

Основной подход, который люди используют для взаимодействия с информационной системой, состоит в том, чтобы рассматривать ее как хранилище данных CRUD(создание, чтение, модификация, удаление). Под этим подразумевается, что у нас есть модель некоторой структуры записей, где мы можем создавать новые записи, читать записи, обновлять существующие записи и удалять записи. В простейшем случае все наши взаимодействия связаны с хранением и извлечением этих записей.

По мере того, как потребности бизнес функций растут, постепенно происходит отдаление от этой модели. Может потребоваться взглянуть на информацию в том виде, в котором она не хранится в хранилище записей. Возможно требуется сложить несколько записей в одну или сформировать виртуальные записи путем объединения информации из разных мест. Что касается обновления, то появляются бизнес-функции проверки, которые позволяют хранить только определенные комбинации данных, или подтвердить, что данные сохранились.

Изменение, которое вводит CQRS, состоит в том, чтобы разделить эту концептуальную модель на отдельные модели для обновления и отображения, которые соответственно называются команда и запрос. Логическое обоснование этого изменения состоит в том, что для многих проблем наличие одинаковой концептуальной модели для команд и запросов приводит к более сложной модели, которую становится сложно поддерживать и изменять.

I. Основные понятия CQRS

К основным понятиям CQRS относятся: команды, командная шина, обработчик команд, шина событий, обработчик событий и запросы.

Команды представляют намерение пользователя. Они содержат всю необходимую информацию о действиях, которые пользователь хотел бы выполнить.

Командная шина – это тип очереди, которая получает команды и передает их обработчикам команд.

Обработчики команд содержат актуальную бизнес-логику, которая проверяет и обрабатывает данные, полученные в командах. Обработчики команд отвечают за генерацию и распространение событий домена на шину событий.

Шина событий отправляет события обработчикам событий, подписанным на определенные типы событий. Шина событий может распространять события как асинхронно, так и синхронно.

Обработчики событий отвечают за обработку определенных входящих событий. Их роль заключается в том, чтобы сохранять новое состояние приложения в репозитории чтения и выполнять действия терминала, такие как отправка электронных писем, хранение файлов и т. д.

Запросы – это объекты, которые представляют фактическое состояние приложения, доступное для пользователя. Получение данных для пользовательского интерфейса должно осуществляться через эти объекты.

II. Источник событий

Источник событий(Event Sourcing) – это специализированный шаблон для хранения данных. Вместо сохранения текущего состояния объекта каждое изменение состояния сохраняется как отдельное событие, которое имеет смысл для бизнес-пользователя.[2] Текущее состояние

рассчитывается путем применения всех событий, которые изменили состояние объекта. С точки зрения CQRS, сохраненные события – это результаты выполнения команды для агрегата на этапе записи. Сторона чтения может обрабатывать события которые передаёт хранилище событий и создавать целевые наборы данных, необходимые для запросов. Ключевые причины для реализации этого шаблона:

- Сохраненные события становятся полным перечнем контрольных изменений, внесенных в приложение. По этой причине источник событий традиционно реализуется приложениями с серьезными потребностями аудита. Это также облегчает отладку;
- Поскольку каждое событие сохраняется, можно создавать отчеты в любое время. Эти отчеты могут даже показывать поэтапно, что произошло в определенный момент времени в прошлом. Это облегчает применение специальной аналитики к данным, в том числе с использованием алгоритмов искусственного интеллекта. По тем же причинам можно гибко изменять сторону чтения приложения, а также интегрироваться с новыми внешними системами;
- Так как получение событий означает только добавление новых данных в хранилище, запись выполняется максимально быстро.

Как видно из перечня, Event Sourcing и CQRS очень тесно связаны и действительно поддерживают друг друга.

III. Преимущества

Можно выделить много преимуществ подхода CQRS, включая следующие:

- Разделение задач по разработке между более опытным персоналом, которые будут работать над бизнес-логикой, и теми, кто будет работать над запросами. Однако, это преимущество может повредить передаче знаний;
- Возможность добиться высокой производительности операций чтения / записи, масштабируя команды и запросы на нескольких разных серверах;[3]
- Использование двух разных репозиториев (чтение / запись), которые синхронизированы, дает автоматическое резервное копирование без каких-либо дополнительных усилий;
- Операции чтения не нагружают базу данных используемую для записи, поэтому они могут работать быстрее при использовании источник событий;
- Структурирование считанных данных непосредственно для представлений, что упрощает представления и повышает производительность.

IV. Недостатки

У шаблонов распределения ответственности по запросам команд и источник событий есть ряд недостатков:

- Сложность: Вся система с каждым компонентом, основанным на источнике событий, сделает взаимодействие между ними сложным и трудным для чтения, если будет не достаточно проработан каждый компонент системы. Если все функциональные возможности влияют на один и тот же источник событий, он быстро превратится в монолитный источник событий. В целом, эти шаблоны добавляют значительную сложность, следует учитывать, стоит ли оно того;
- Повышенное использование дискового пространства: хранилище событий может в конечном итоге использовать много дискового пространства для хранения событий.[4]

V. Вывод

Источник событий выгодно использовать в программных системах, где бизнес может извлечь выгоду из истории событий, которые произошли. Одним из ключевых преимуществ источника событий является расширяемость программных систем – когда необходимо добавить новый компонент отчетности, можно просто воспроизвести на нем исторические события и запустить его. Это очень выгодно в сине-зеленых развертываниях, когда необходимо чтобы приложения не простаивало.

Интеграция с внешними системами может быть выполнена с использованием событий. В таких сценариях события – это наш программный интерфейс приложения, и внешняя система должна их понимать. Конечно, мы должны публиковать только те события, которые являются публично важными.

Список литературы

1. A website on building software effectively. [Электронный ресурс] / CQRS – Режим доступа: https://martinfowler.com/bliki/CQRS.html. – Дата доступа: 06.10.2019.
2. Framework and server for event-driven microservices focused on CQRS and Event Sourcing. [Электронный ресурс] / Event Sourcing with Axon – Режим доступа: https://axoniq.io/resources/event-sourcing. – Дата доступа: 06.10.2019.
3. RisingStack, Full Stack Javascript, Node.js Development and Consulting [Электронный ресурс] / When to use CQRS?! – Режим доступа: https://community.risingstack.com/when-to-use-cqrs. – Дата доступа: 05.10.2019.
4. Benefits and drawbacks of using CQRS and Event Sorucing patterns on real life example. [Электронный ресурс] / CQRS and Event Sourcing as an antidote for problems with retrieving application states – Режим доступа: https://www.nexocode.com/blog/posts/cqrs-and-event-sourcing. – Дата доступа: 06.10.2019.