DOI 10.37882/2223–2966.2020.05.38

ИСПОЛЬЗОВАНИЯ ШАБЛОНА «САГА» ДЛЯ ПОДДЕРЖАНИЯ СОГЛАСОВАННОСТИ ДАННЫХ В МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ

USING "SAGA" PATTERN FOR PROVIDING CONSISTENCY IN MICROSERVICE ARCHITECTURE

D. Rubtsov

Summary. This article is devoted to the study using the "Saga" pattern building microservice architecture. "Saga" allows implementing asynchronous distributed transactions, which helps keep the state between microservice eventually consistent. There are several methods of implementing this pattern described in this article, and the approach for atomic broker message publication is also being studied. Without which it's not possible to achieve complete consistency using "Sagas".

Keywords: microservice architecture, distributed transactions, saga.

Рубцов Дмитрий Викторович
Руководитель группы разработки, Яндекс, г. Москва
rubtsov.dmv@gmail.com

Аннотация. Статья посвящена исследованию использования шаблона «Сага» при проектировании микросервисной архитектуры. Шаблон «Сага» позволяет реализовать асинхронную распределенную транзакцию, с помощью которой сохраняется согласованность по данным между разными микросервисами. В статье рассматриваются различные методы реализации данного шаблона, а также изучается подход к атомарной публикации событий в брокер сообщений, без которого в данном шаблоне нельзя достичь полной согласованности данных.

Ключевые слова: микросервисная архитектура, распределенные транзакции, сага.

Введение

В современных распределенных системах все чаще применяют микросервисную архитектуру. Основная идея микросервисной архитектуры заключается в разделении приложения на более мелкие сервисы, называемые микросервисами, которые общаются между собой с помощью механизма удаленного вызова процедур (RPC — remote procedure call) по сети TCP/IP. Данная архитектура позволяет делать систему более гибкой и масштабируемой [1, с. 10]. Как правило, каждый микросервис имеет свою собственную базу данных (БД), где храниться состоянии объектов предметной области, за которые отвечает данный сервис. Рассмотрим пример, представим, что имеется система по созданию заказов в магазине. Рассмотрим процесс оформления заказа, если бы данная система была монолитной (рисунок 1). При обработке запроса оформления заказа создается транзакция в БД, далее обновляется значение остатка денежный средств на счету в таблице покупателя, затем обновляется статус заказа в таблице с заказами, после чего происходит завершение транзакции и изменения применяются. Если какой-либо запрос не будет выполнен все изменения в БД будут отменены, если БД удовлетворяет требованиям атомарности транзакций.

Рассмотрим тот же самый процесс только в микросервисной архитектуре. Предположим, что у нас есть отдельно сервис покупателей, где хранится вся информация о покупателях, в том числе остаток на лицевом счете, а информация о всех заказах хранится в сервисе заказов. Тогда для оформления заказа требуется выполнить сначала запрос в сервис покупателей и уменьшить остаток средств на лицевом счете, затем сделать запрос в сервис заказов и обновить статус заказа (рисунок 2).

Таким образом бизнес-транзакция на оформления заказа стала распределенной между несколькими БД. Став распределенной, транзакция перестала удовлетворяет требованиям атомарности и изоляции. Если запрос на обновления статуса не будет выполнен, то баланс на счете покупателя не вернется в предыдущее значение. При этом, если в момент выполнения запроса на оформления заказа придет запрос на получение баланса покупателя, а запрос на обновления заказа еще не выполнился, то система вернет уже обновленный баланс, хотя заказ еще не оформлен. Таким образом система приходит в несогласованное состояние.

Одним из возможных решений данной проблем является шаблон «Сага».

Шаблон «Сага»

Суть шаблона «Сага», заключается в разбиении распределенной транзакции на последовательность локальных транзакций [3, с. 250]. При обновлении базы данных каждая локальная транзакция публикует сообщение, которое провоцирует выполнение следующей транзакции. Если локальная транзакция по каким-либо причинам

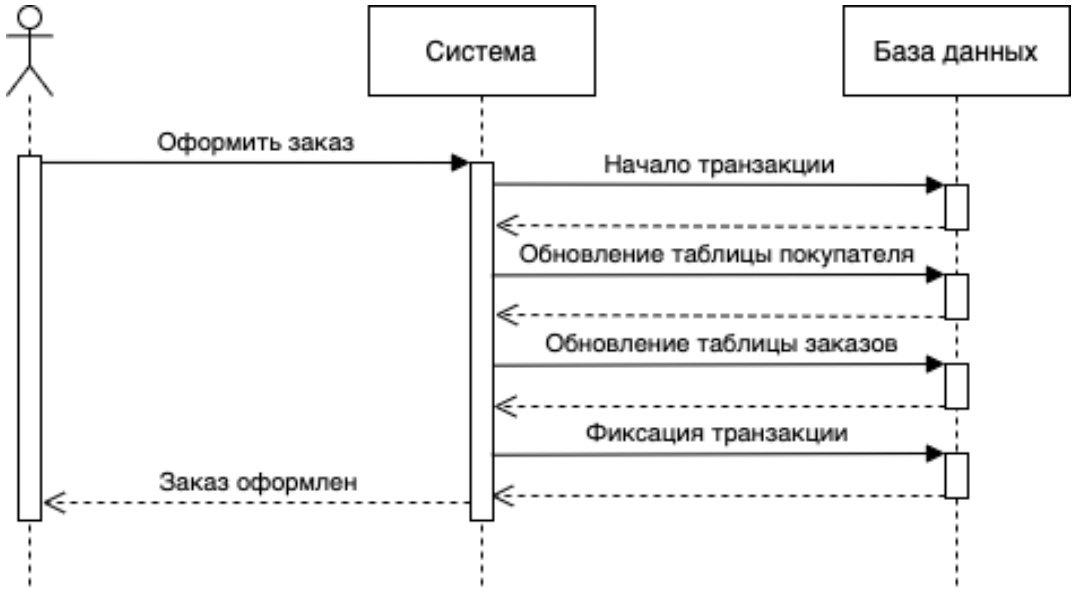

Рис. 1. Процесс оформления заказа в монолитной архитектуре

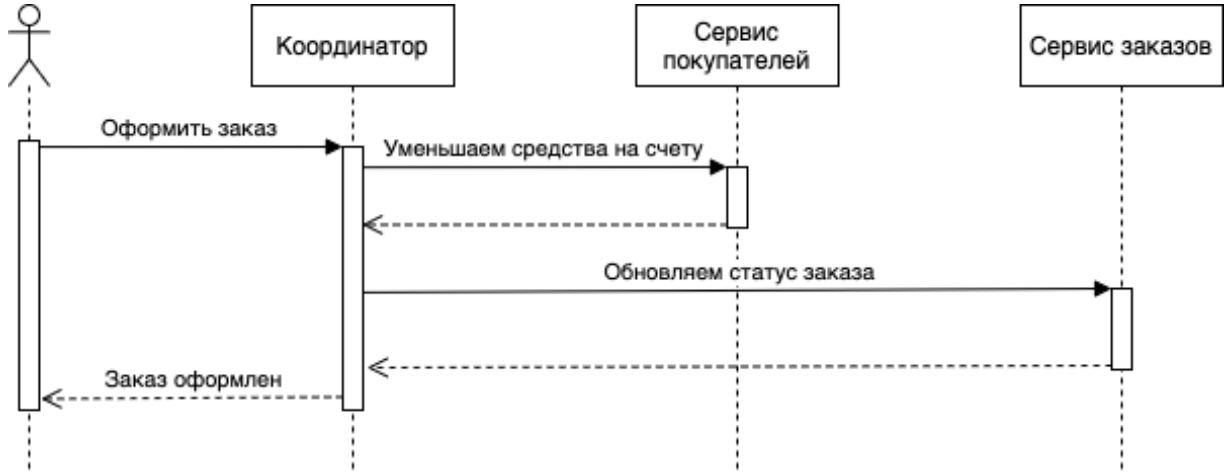

Рис. 2. Процесс оформления заказа в микросервисной архитектуре

не была выполнена, например баланс лицевого счета пользователя ушел в минус, запускается цепочка компенсирующих локальных транзакций, которые откатывают произошедшие изменения в предыдущих транзакциях.

Существует 2 метода координирования выполнения локальных транзакций в «Сагах»:

- Оркестрация
- Хореография

Оркеструемая «Сага»

В оркеструемой «Саге» существует объект оркестратор, который следит за выполнением каждой транзакции и запускает следующие локальные транзакции или откатывает их с помощью компенсирующих транзакций. Этот объект может быть реализован как часть сервиса, в который приходит изначальный запрос, либо может быть отдельной сущностью.

Рассмотрим все тот же пример с оформлением заказ, но уже с использованием «Саги» (рисунок 4). При поступлении запроса в систему координатор создает «Сагу», которая выполняется асинхронно. При выполнении транзакции, координатор отправляет сообщение в сервис заказов и переводит статус заказа в «Оформляется», далее выполняется запрос в сервис покупателей, где снимаются средства со счета покупателя, «Сага» завер-

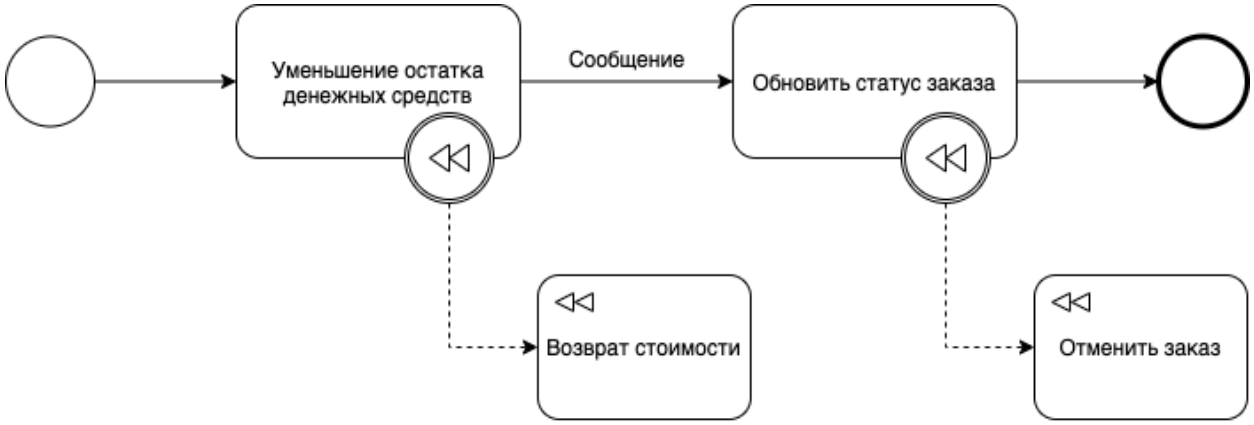

Рис. 3. «Сага» процесса оформления заказа

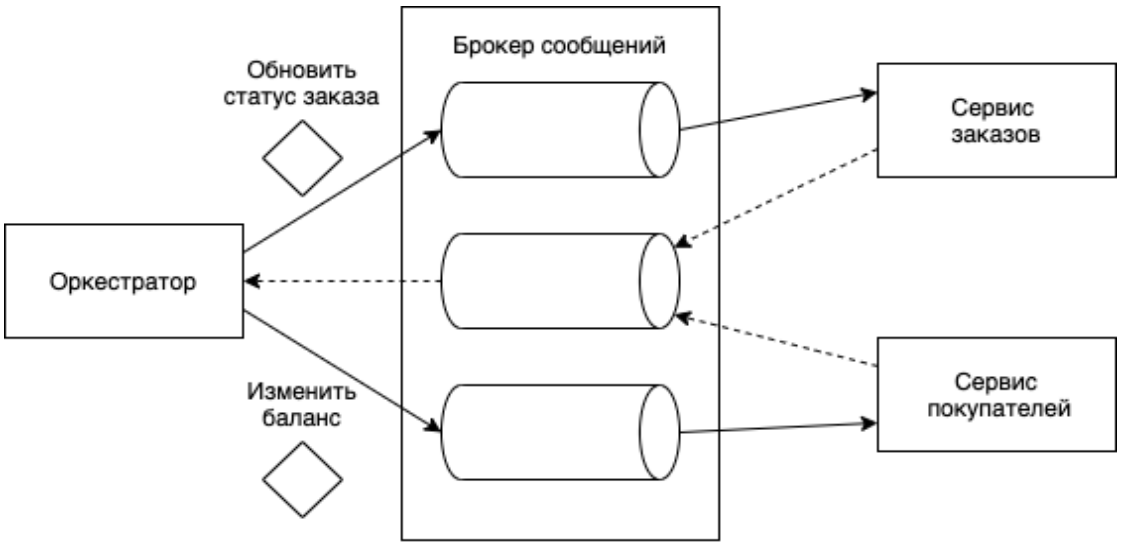

Рис. 4. Архитектура оркеструемой «Саги»

шается транзакцией, которая оформляет статус заказа в «Оформлен».

Хореографическая «Сага»

В хореографическое «Саге» каждая локальная транзакция публикует событие, а сервисы, которые следуют за этой транзакцией самостоятельно подписываются на данное событие, при получении события выполняют свою локальную транзакцию и публикуют свое событие. Если транзакция не была выполнена, сервис отправляет событие об отмене транзакции, которое получает предыдущий сервис и откатывает свою транзакцию и также публикует событие, продолжая цепочку из компенсирующих транзакций. Для обмена событиями обычно используются брокеры сообщений.

Рассмотрим пример с оформлением заказа (рисунок 5). Сервис заказов переводит заказ в статус «Оформляется» и публикует событие об оформлении, сервис покупателей получает это событие и снимает средства с лицевого счета, и публикует событие об успешном снятии средств, сервис заказа получает это сообщение и переводит заказ в статус «Оформлен». Данный метод позволяет распараллелить выполнение локальных транзакций между микросервисами.

Обработка ошибок

В системе при выполнении локальных транзакций могут возникать два типа ошибок: ошибки бизнес логики и технические ошибки. Ошибки бизнес логики возникают, когда локальную транзакцию невозможно выполнить из-за несоблюдения бизнес правил, например, не хватает

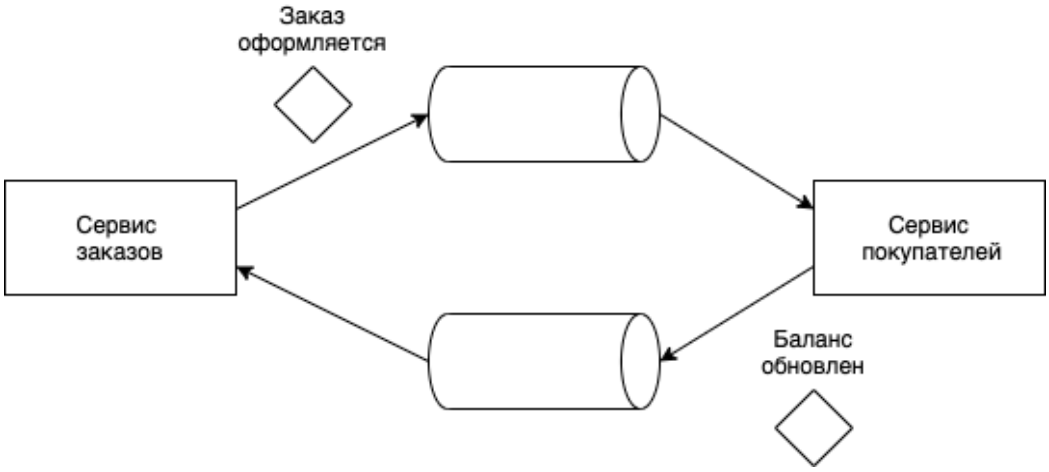

Рис. 5. Архитектура хореографической «Саги»

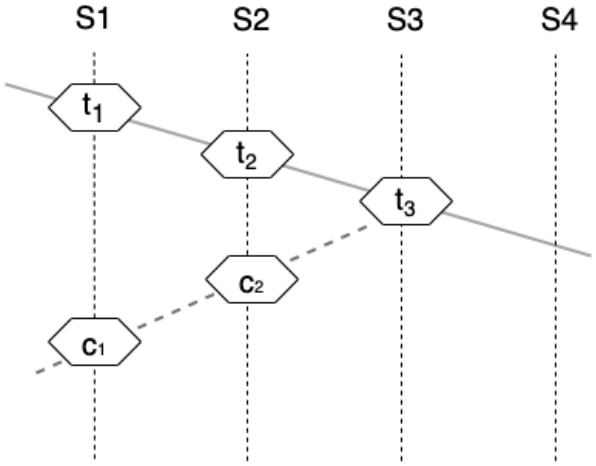

Рис. 6. Обработка ошибки при выполнении «Саги»

средств на счете клиента. Подобные ошибки должны сразу приводить к запуску цепочки компенсирующих транзакций. Рассмотрим пример, допустим имеется бизнес-транзакция, которая распределена между 4 микросервисами с локальными транзакциями. Таким образом «Сага» состоит из множества локальных транзакций $T = \{t_i | i \in [1;4]\}$, t_i — локальная транзакция в сервисе S_i и множества $C = \{c_i | i \in [1;4]\}$, c_i — локальная компенсирующая транзакция в сервисе S_i, после выполнения которой все изменения произведены в соответствующей транзакции t_i должны быть отменены [2, с. 153]. Предположим, что данная «Сага» не смогла выполнить транзакцию t_3, далее координатор, в случае оркеструемой саги, должен последовательно выполнить транзакции c_2, c_1 (рисунок 6).

Второй тип ошибок — технический, связан с тем, что все общение между микросервисами осуществляется по сети, где могут возникать временные проблемы с доступностью. Эти ошибки неизбежно будут возникать в распределенных системах и именно поэтому все транзакции, как прямые, так и компенсирующие, должны быть идемпотентными. Т.е. необходимо реализовать данные транзакции таким образом, чтобы их можно было повторно выполнить и получить тот же результат. Если условие идемпотентности выполняется, то при возникновении временных технических ошибок необходимо повторять выполнение транзакции, конечно используя механизмы сглаживание нагрузки, т.к. ошибка может возникать из-за перегрузки конкретного микросервиса. Если по каким-либо причинам при откате компенсирующая транзакция не может выполниться после определенного количества повторений, данную «Сагу» стоит пометить как «сломанную» и прекратить ее выполнение. Также рекомендуется настроить мониторинг на подоб-

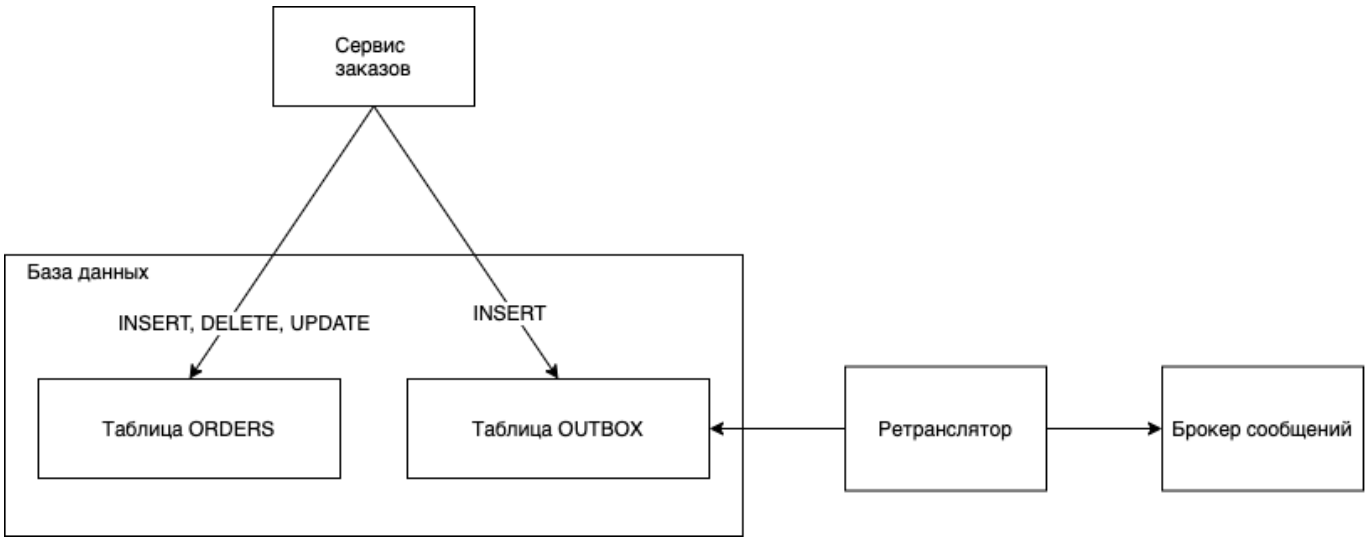

Рис. 7. Шаблон «Публикация событий»

ные «Саги», так как для их разрешения требуется ручное вмешательство разработчика. Чтобы не допускать подобных ситуации в реальной среде требуется покрыть автоматическими тестами каждую транзакцию, включая компенсирующие транзакции.

Атомарная поставка сообщений в брокер

Так как в шаблоне «Сага» для передачи событий между микросервисами используется брокер сообщений. Как правило, брокер сообщений — это отдельный сервис, отличный от основной базы данных, например RabbitMQ, Apache Kafka, Amazon SQS и другие. Для того, чтобы согласованность данных в системе сохранялась, публикация сообщений в очередь должна быть согласована с локальной транзакцией, которая приводит к отправке сообщения в брокер. Если сообщение в очередь отправляется до завершения локальной транзакции, то сообщение отправится, хотя нет гарантии, что фиксирование транзакции в локальной БД выполнится успешно. В случае отката локальной транзакции, сообщение все равно попадет в очередь, что приведет к несогласованному изменению в следующем сервисе. Если отправлять сообщение после фиксации транзакции, то есть обратный риск, того, что транзакция будет зафиксирована, но сообщение не получится отравить, например, из-за временных проблем с доступностью.

Для решения данной проблемы существует шаблон «Публикация событий» [2, с. 133], его суть заключается в том, что в локальной БД создается таблица OUTBOX, которая играет роль локальной очереди. Вместо отправки сообщения напрямую в брокер, обработчик запроса записывает событие в данную таблицу в одной транзакции с изменением бизнес-объектов. А отдельный процесс, называемый «ретранслятором», следит за новыми записями в таблице OUTBOX и гарантированно отправляет данные сообщение в брокер. Таким образов в брокере гарантированно окажутся только те события, которые были полностью применены в локально базе данных.

Но при использовании некоторых баз данных, с целью упрощения логики обработчика запросов, при условии, что основная часть транзакций приводит к публикации события, можно ретранслятор настроить на прямое чтение лога транзакций самой БД. В большинстве баз данных реализован свои механизм для чтения лога транзакций, например, в случае MySQL — это binlog, а в случае PostgreSQL — это WAL.

Преимущества и недостатки использования «Саг»

Основным преимуществом данного подхода является то, что в отличии от двухфазного коммита, транзакция выполняется асинхронно, что особенно важно, когда количество микросервисов под одной бизнес транзакцией достаточно большое. Но при этом подходе систему становится гораздо сложнее тестировать и отлаживать. Важной особенностью является еще и то, что поскольку транзакция выполняется асинхронно, то согласованность наступает через какое-то время после начала транзакции. Таким образом при распределенных транзакциях реализованных с помощью «Саг» не выполняется условие изоляции, т.е. промежуточные результаты транзакций будут видны из других транзакций, что тоже

накладывает свои ограничения и вносит дополнительную сложность в реализации.

Заключение

Таким образом, использование «Саг» позволяет сохранять согласованность в системах с большим количеством микросервисов, характерные для систем, разрабатываемых в крупных компаниях. Данный подход является одним из наиболее привлекательных методов для реализации распределенных транзакций в микросервисной архитектуре. Однако нужно учитывать, что транзакции, реализованные данным способом, не удовлетворяют требованиям изоляции транзакций, и использование данного подхода требует адаптации процесса отладки и тестирования.

ЛИТЕРАТУРА

1. Ньюмен С. Создание микросервисов. Спб.: Питер, 2016. 304 с.
2. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. Спб.: Питер, 2019. 544 с.
3. Garcia-Molina H., Salem K. SAGAS. SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data. New York, United States: Association for Computing Machinery, 1987. pp. 249–259.

Журнал «Современная наука: актуальные проблемы теории и практики»

Яндекс, здание офиса в Москве