

Еще про проектирования брокеров сообщений.

Мы с вами затронули главные концепции работы брокеров сообщений. Теперь вы умеете выбирать модель взаимодействия (очередь/подписка), модель взаимодействия брокера-консьюмера (push/pull) и гарантию доставки (не более одного раза, не менее 1 раза, строго 1 раз). Исходя из требований к вашему приложению вы принимаете решение, и останется выбрать подходящего брокера, совместно с командой разработки.

Сейчас же рассмотрим реализацию двух паттернов обмена данными через брокеров. Т.е. на более высоком уровне рассмотрим систему, помня, какие у нас есть концепции работы.

Запрос-ответ (Request-Response)	Односторонний поток (One-Way)
Используется, когда одна система должна ожидать ответа от другой и не должна продолжать работу.	Используется, когда нам не нужен ответ от потребителя.
Реализуем синхронный бизнес-процесс.	Реализуем асинхронный бизнес-процесс.
Продюсер контролирует работу консьюмера. В случае неприемлемого ответа может отправить запрос заново.	Продюсер никак не контролирует работу консьюмера, т.к. не участвует в логике бизнес-процесса после отправки сообщения.

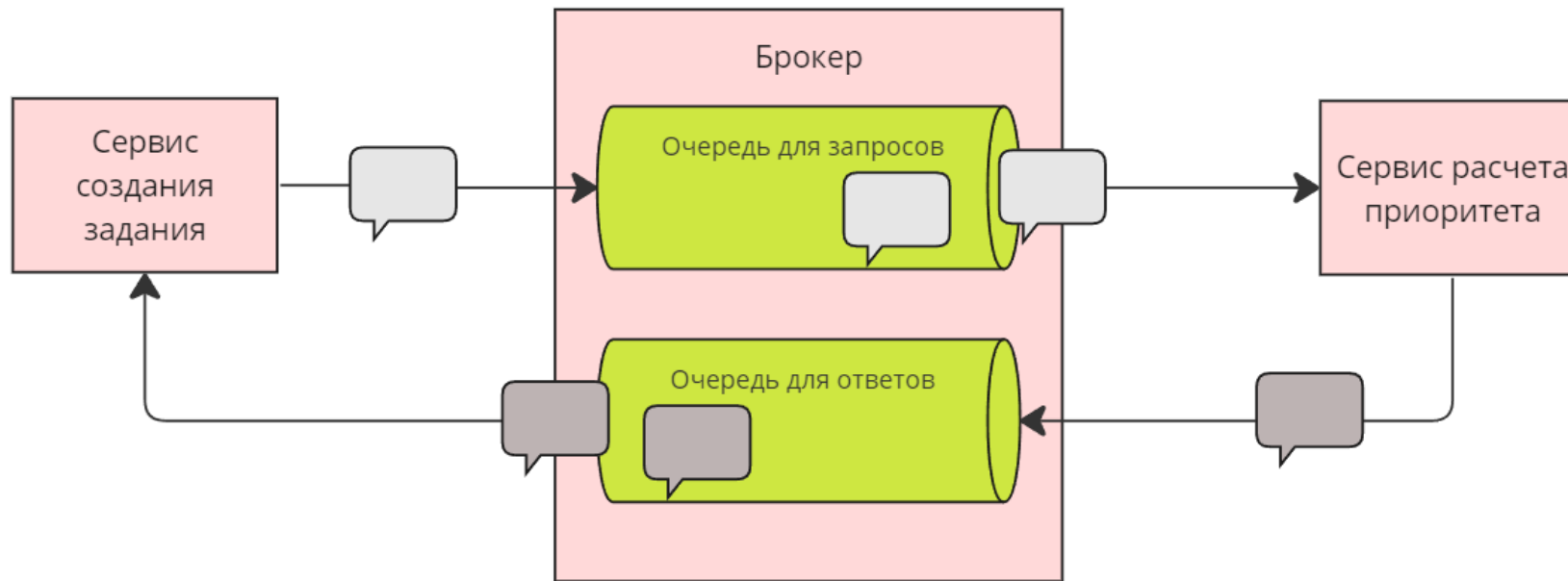
Ранее в наших примерах мы рассматривали односторонний поток сообщений. Продюсер отправлял сообщение, и ему не важно, как его обработает консьюмер. Могут быть важны лишь гарантии доставки. Поэтому останавливаться на этом паттерне не будем. Но теперь представьте, что вам нужно реализовать синхронный бизнес-процесс. Как это сделать?

Допустим, есть два компонента приложения, сервис создания задания и сервис расчета приоритета задания. В процессе создания задания мы обращались в сервис расчета по HTTP-запросу. И по причине долгих ответов от сервиса расчета приоритета принято решение реализовать запрос-ответ с помощью брокеров. В таком случае избегаем долгих HTTP-запросов и получаем повышенную надежность,

уменьшая связность между компонентами. Если сервис расчета приоритета "упадет", сервис создания заказа будет ожидать ответа.

Тут стоит внимать в детали. Техническое взаимодействие будет асинхронным, так как мы используем брокер сообщений. Но сам бизнес-процесс - синхронным, т.к. сервис создания задания не может продолжить работу, создать полноценное задание, без получения информации о приоритете от сервиса расчета приоритета.

Схематично это будет выглядеть так:



Во-первых, вы заметили, что брокер может содержать несколько очередей (также и топиков). Ранее мы с вами не затрагивали эту тему, но конечно, брокер сообщений может содержать больше одной очереди/топика. Сервисов для интеграции может быть много, а также и бизнес-типов сообщений может быть много, каждый тип нужно доставить конкретным получателям.

В данном случае мы сделали две очереди. В одну мы принимаем сообщения от сервиса создания заданий, и консьюмером работает сервис расчета приоритета. Когда он рассчитал приоритет, отправляет сообщение во вторую очередь (для ответов), и консьюмером уже является сервис создания, который при получении сообщения из второй очереди завершает создание задания. В случае, если ответ сервису создания не понравится (допустим, он получит null), он может еще раз отправить сообщение в сервис расчета. Все зависит от бизнес-логики, которую мы заложим.

Во-вторых, мы выделили сообщения разным цветом. Для правильной реализации такого паттерна нужно:

- Чтобы сообщения содержали ID системы, брокер должен понимать, в какую очередь отправить сообщение
- Чтобы все сообщения содержали уникальный ID для сопоставления запроса-ответа (припоминаете JSON-RPC?)

По итогу сообщения отличаются по ID системы но запрос/ответ совпадают по уникальному ID.

Мы разобрали, как вы можете реализовать синхронный бизнес-процесс с помощью брокеров сообщений. Важно понимать, зачем вы хотите использовать такой подход и какие плюсы вы получите. Часто такой подход используют для уменьшения связности между приложениями, особенно, если какое-либо приложение уже legacy (не будет дорабатываться, устаревшее). Но появляются и свои минусы. Например, меньшая пропускная способность, если заданий будет очень много, и появление точки отказа в виде брокера сообщений, нужно контролировать его стабильность.