**JMS (Java Message Service)** - это стандарт обмена сообщениями, который используется для отправки и получения сообщений очередями. Этот процесс позволяет вести распределенный обмен сообщениями с внешними системами надежно, асинхронно и без жесткой фиксации по парам.

JMS предоставляет асинхронный и надежный способ отправки и получения сообщений между клиентами, используя промежуточный посредник (message broker) для обработки сообщений.

Программные компоненты взаимодействуют косвенно. Компонент-отправитель не требует знаний о компоненте получателе. Отправитель положил сообщение в очередь, и забыл. Ему по сути все равно кто его получит. Получатель может даже не работать на момент отправки сообщения в очередь. Когда получатель запустится, он заберет сообщение из очереди, а до тех пор оно будет хранится там.

JMS поддерживает **две модели** обмена сообщениями: **PTP или «точка-точка»** и «**публикация-подписка**». Первая модель (PTP) опирается на концепцию очередей сообщений, где ключевыми характеристиками являются один потребитель на сообщение и отсутствие временной зависимости между отправителем и получателем. Модель публикации-подписки — наоборот: несколько потребителей на сообщение и временная зависимость между отправителем и получателем.

В веб-приложениях очереди часто используются для отложенной обработки событий или в качестве временного буфера между другими сервисами, тем самым защищая их от всплесков нагрузки.

**Основные компоненты** JMS:

1. **Producer** (производитель): Клиент, который отправляет сообщения в JMS очередь или тему. Он создает и отправляет сообщения определенному адресату.
2. **Consumer** (потребитель): Клиент, который получает сообщения из JMS очереди или темы. Он подписывается на адресата и получает сообщения, когда они становятся доступными.
3. **Message** (сообщение): Информация, которая передается между Producer и Consumer. Сообщение может содержать различные типы данных, такие как текст, числа, объекты и т.д.
4. **Queue** (очередь): Механизм доставки сообщений, где сообщения хранятся и обрабатываются в порядке их поступления. Каждое сообщение в очереди доставляется только одному Consumer.
5. **Topic** (тема): Механизм доставки сообщений, где сообщения отправляются всем подписчикам (Consumer), которые подписались на эту тему. Каждое сообщение в теме доставляется всем подписчикам.

Примеры популярных JMS посредников (реализации) включают Apache ActiveMQ, IBM MQ, RabbitMQ и другие.

**Для начала работы** с JMS API необходимо выполнить следующие шаги:

1. Создать соединение с JMS провайдером.
2. Создать сессию для отправки и получения сообщений.
3. Создать объекты для отправки и получения сообщений (Destination, MessageProducer и MessageConsumer).
4. Отправить сообщение и/или получить сообщение.

Для создания соединения с JMS провайдером используйте объект *ConnectionFactory.*

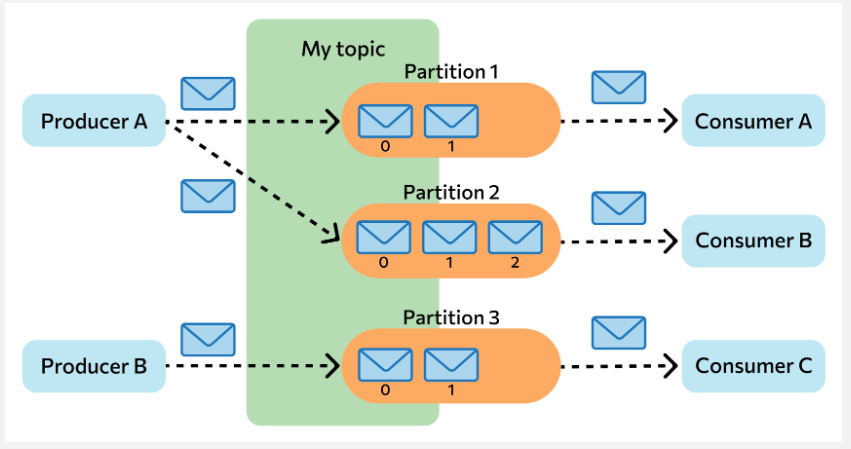
**Kafka Apache** — распределенная система обмена сообщениями между серверными приложениями в режиме реального времени.

Ключевые концепции:

* **Producer –** приложение или процесс, публикующий сообщение
* **потребитель (consumer)** - приложение или процесс, который принимает сгенерированное продюсером сообщение;
* **сообщение** — пакет данных, необходимый для совершения какой-либо операции (например, авторизации, оформления покупки или подписки);
* **брокер** — узел (диспетчер) передачи сообщения от процесса-продюсера приложению-потребителю;
* **топик (тема)** — виртуальное хранилище сообщений (журнал записей) одинакового или похожего содержания, из которого приложение-потребитель извлекает необходимую ему информацию.

Работа Kafka Apache выглядит следующим образом:

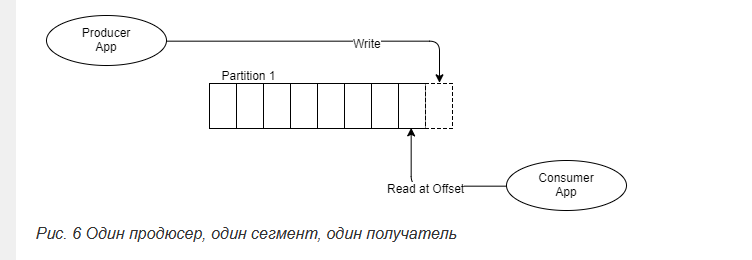
* Приложение-продюсер создает сообщение и отправляет его на узел Kafka.
* Брокер сохраняет сообщение в топике, на который подписаны приложения-потребители.
* Потребитель при необходимости делает запрос в топик и получает из него нужные данные.



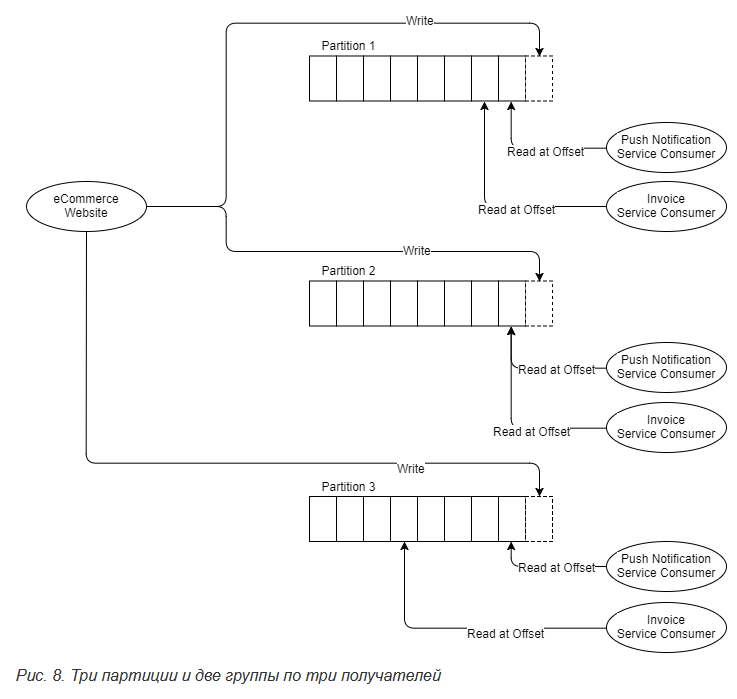
Сообщения хранятся в Kafka в виде журнала коммитов — записей, размещенных в строгой последовательности. Их можно только добавлять. Удалять или корректировать нельзя. Сообщения хранятся в той последовательности, в которой поступили, их считывание ведется слева направо. Хранение может длиться в течение определенного периода или до достижения заданного порога.

Топик обычно разделен на партиции, а потребители на группы. При этом одновременно только один потребитель из группы может читать партицию (Потребитель может читать и несколько партиций, но у каждой партиции может быть только один потребитель), таким образом распределяется нагрузка и предотвращается повторное чтение данных. И если потребителей больше, чем партиций, то лишние потребители вообще не будут задействованы в обработке сообщений.

Каждое сообщение партиции имеет свой собственный уникальный монотонно возрастающий offset. Потребитель при чтении сообщает брокеру какой offset уже был прочитан, и при последующих подключениях брокер возвращает это значение, чтобы начать чтение с нужной позиции.



Потребители из разных групп могут читать партиции одновременно. При этом консьюмеры, находящиеся в разных группах могут иметь различные указатели на уже прочитанные данные.



Мы можем также переместится назад относительно offset, и еще раз прочитать ранее прочитанные сообщения. Это может быть полезно при восстановлении после сбоев.

Разделять топик на партиции можно по какому либо условию, что может улучшить параллельную обработку.

Партиции невозможно удалить после их создания, можно удалить только весь топик целиком.

Преимущества Kafka:

* Отказоустойчивость – узлы Kafka могут содержатся на нескольких кластерах. Принимая сообщение от продюсера, она реплицирует (копирует) его, а копии сохраняет на разных узлах. При этом один из брокеров назначается ведомым в секции, через него потребители будут обращаться к записям. Другие брокеры остаются ведомыми, их главная задача — обеспечить сохранность сообщения (его копий) даже при выходе одного или нескольких узлов из строя.
* Масштабируемость – можно легко добавлять новые машины в кластер, не отключая всю систему.
* Производительность - процессы генерирования/отправки и считывания сообщений организованы независимо друг от друга. Тысячи приложений, процессов могут одновременно и параллельно играть роль генераторов и потребителей сообщений.
* Безопасность – можно настроить уровень изоляции для транзакций, можно исключить чтение незавершенных или отмененных сообщений.
* Долговечность - Данные в Kafka сохраняются в долговременные виртуальные хранилища в течение заданного периода времени (дней, недель, месяцев). За счет распределенного хранения информации она не потеряется при сбое одного или нескольких узлов

**RabbitMQ**

**AMQP (Advanced Message Queuing Protocol) —** открытый протокол для передачи сообщений между компонентами системы. Основная идея состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет маршрутизацию, возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений.

Протокол **AMQP** вводит три понятия:

* **exchange** (обменник или точка обмена) — в неё отправляются сообщения. Обменник **распределяет сообщение** в одну или несколько очередей. Он **маршрутизирует сообщения в очередь** на основе созданных связей (binding) между ним и очередью
* **queue (очередь)** — структура данных на диске или в оперативной памяти, которая **хранит ссылки на сообщения и отдает копии сообщений consumers** (потребителям). Одна очередь может использоваться несколькими потребителями
* **binding (привязка)** — правило, которое **сообщает точке обмена в какую из очередей эти сообщения должны попадать**. Обменник и очередь могут быть связаны несколькими привязками

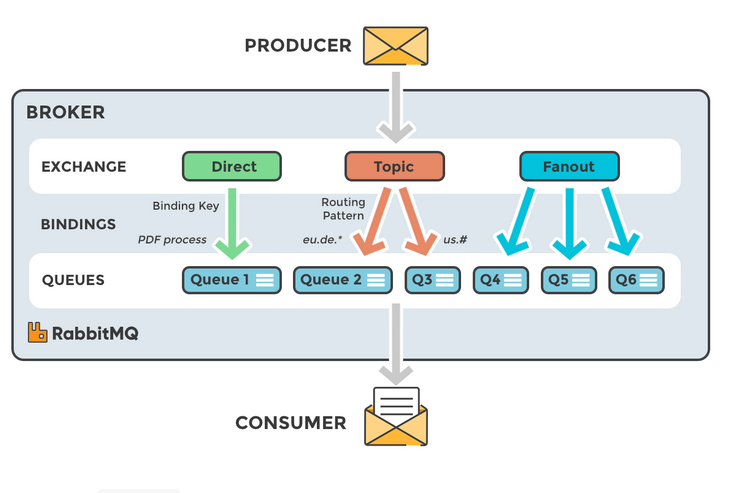
**Где используются очереди:**

* Для асинхронного взаимодействия между между сервисами (Например в микросервисах)

**RabbitMQ –** это брокер сообщений с открытым исходным кодом. Он маршрутизирует собщения по всем базовым принципам протокола AMQP описанным в спецификации.

В RabbitMQ producer не отправляет сообщение непосредственно в очередь. На самом деле и довольно часто издатель даже не знает, будет ли сообщение вообще доставлено в какую-либо очередь.

Вместо этого издатель может отправлять сообщения только на обмен. С одной стороны, обмен получает сообщения от издателей, а с другой — отправляет их в очереди. Обмен должен точно знать, что делать с полученным сообщением. Должно ли оно быть добавлено в определенную очередь? Должно ли оно быть добавлено в несколько очередей? Или сообщение нужно игнорировать.



Кратко работу RabbitMQ можно описать следующим образом:

1. Издатель отправляет сообщение определенному обменнику
2. Обменник, получив сообщение, маршрутизирует его в одну или несколько очередей в соответствии с правилами привязки между ним и очередью
3. Очередь хранит ссылку на это сообщение. Само сообщение хранится в оперативной памяти или на диске
4. Как только потребитель готов получить сообщение из очереди, сервер создает копию сообщения по ссылке и отправляет
5. Потребитель получает сообщение и отправляет брокеру подтверждение
6. Брокер, получив подтверждение, удаляет копию сообщения из очереди. Затем удаляет из оперативной памяти и с диска

**Exchange** — обменник или точка обмена. В него отправляются сообщения. Exchange **распределяет сообщение** в одну или несколько очередей. Он **маршрутизирует сообщения в очередь** на основе созданных связей (bindings) между ним и очередью.

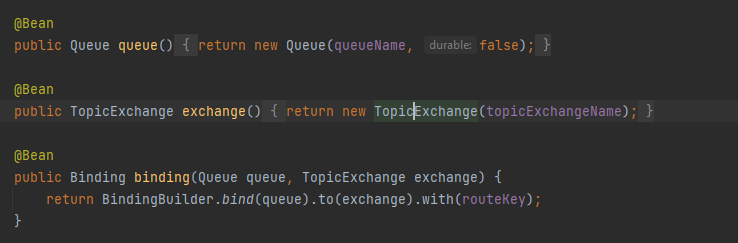
Direct exchange — используется, когда нужно **доставить сообщение в определенные очереди**. Сообщение публикуется в обменник с определенным **ключом маршрутизации** и попадает во все очереди, которые связаны с этим обменником аналогичным ключом маршрутизации. **Ключ маршрутизации — это строка**. Поиск соответствия происходит при помощи **проверки строк на эквивалентность**.

Topic exchange – аналогично direct exchange дает возможность осуществления выборочной маршрутизации путем сравнения ключа маршрутизации. Но, в данном случае, ключ задается **по шаблону**.

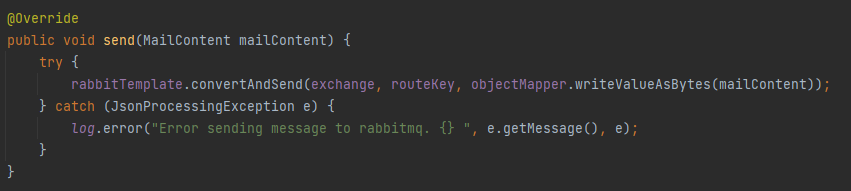
**Queue** (очередь) — структура данных на диске или в оперативной памяти, которая хранит ссылки на сообщения и отдает их копии consumers (потребителям).

**Binding** (привязка) — правило, которое сообщает обменнику в какую из очередей должны попадать сообщения.

Если создание очереди происходит с установленным параметром durable, то такая очередь **сохраняет свое состояние** и восстанавливается после перезапуска сервера/брокера. Данная очередь будет существовать до тех пор пока не будет вызвана команду Queue.Delete.



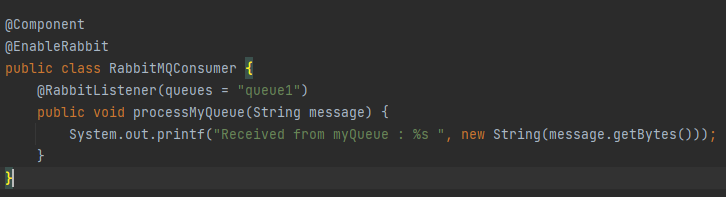
Отправка сообщение в очередь с помощью **RabbitTemplate.**



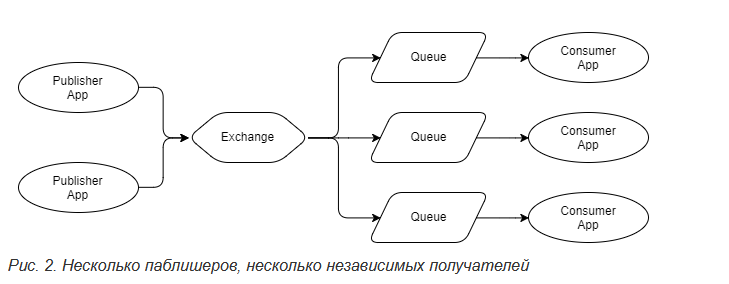
Конфигурация RabbitTemplate:



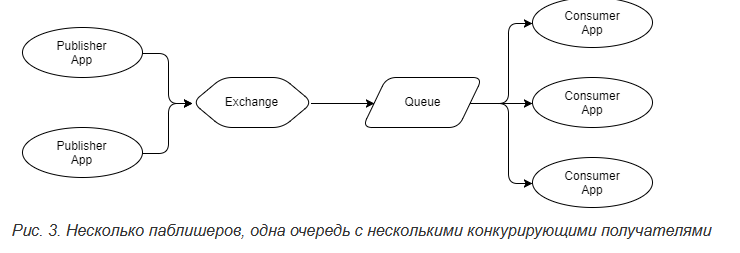
В простейшем случае слушателя можно создать с помощью аннотаций:



Если нужно чтобы несколько потребителей получали сообщение, нужно несколько очередей. Сообщение отправляется в Exchange, а Exchange отправляет его по очередям. Затем слушатели извлекают сообщения, каждый из своей очереди.



Для масштабирования можно использовать конкурирующих получателей. Они будут брать сообщения из одной очереди, и тот кто первым успеет, тот и обработает сообщение.



RabbitMQ позволяет выносить недоставленные сообщения, или сообщения с истекшим TTL в отдельную очередь, где они могут быть обработаны.

**Отличия между Kafka и очередями сообщений**

* Сообщения в Kafka не удаляются брокерами по мере их обработки консьюмерами – данные могут хранится днями, неделями, годами.
* Благодаря этому одно и то же сообщение может быть обработано сколько угодно раз разными консьюмерами, либо даже одним и тем же консьюмером.
* В RabbitMQ когда консьюмер подтверждает, что обработал сообщение, оно удаляется из очереди.
* В RabbitMQ чтобы несколько сервисов могли получать сообщения, нужно иметь несколько очередей (каждому сервису своя очередь).

В Kafka достаточно послать сообщение один раз, а консьюмеры считают его по мере необходимости.

* В Kafka сообщения реплицируются на несколько брокеров, и отказ одного из брокеров не приведет к большим последствиям. В RabbitMQ в случае отказа брокера сообщения могут быть потеряны.
* В RabbitMQ сообщения упорядочены в рамках очереди. В Kafka сообщения не упорядочены в рамках топика, но упорядочены в рамках партиций.