

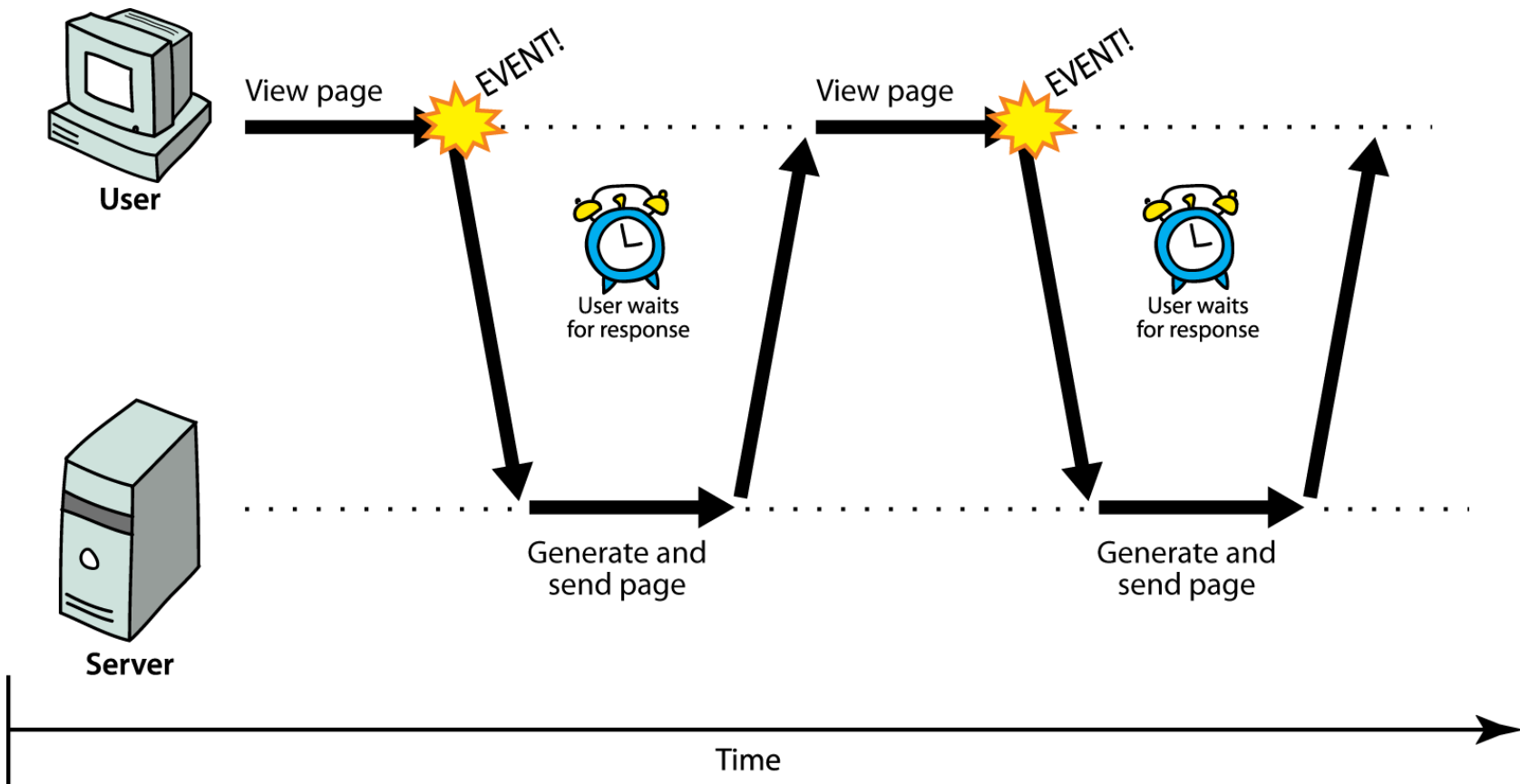


СБЕРБАНК ТЕХНОЛОГИИ

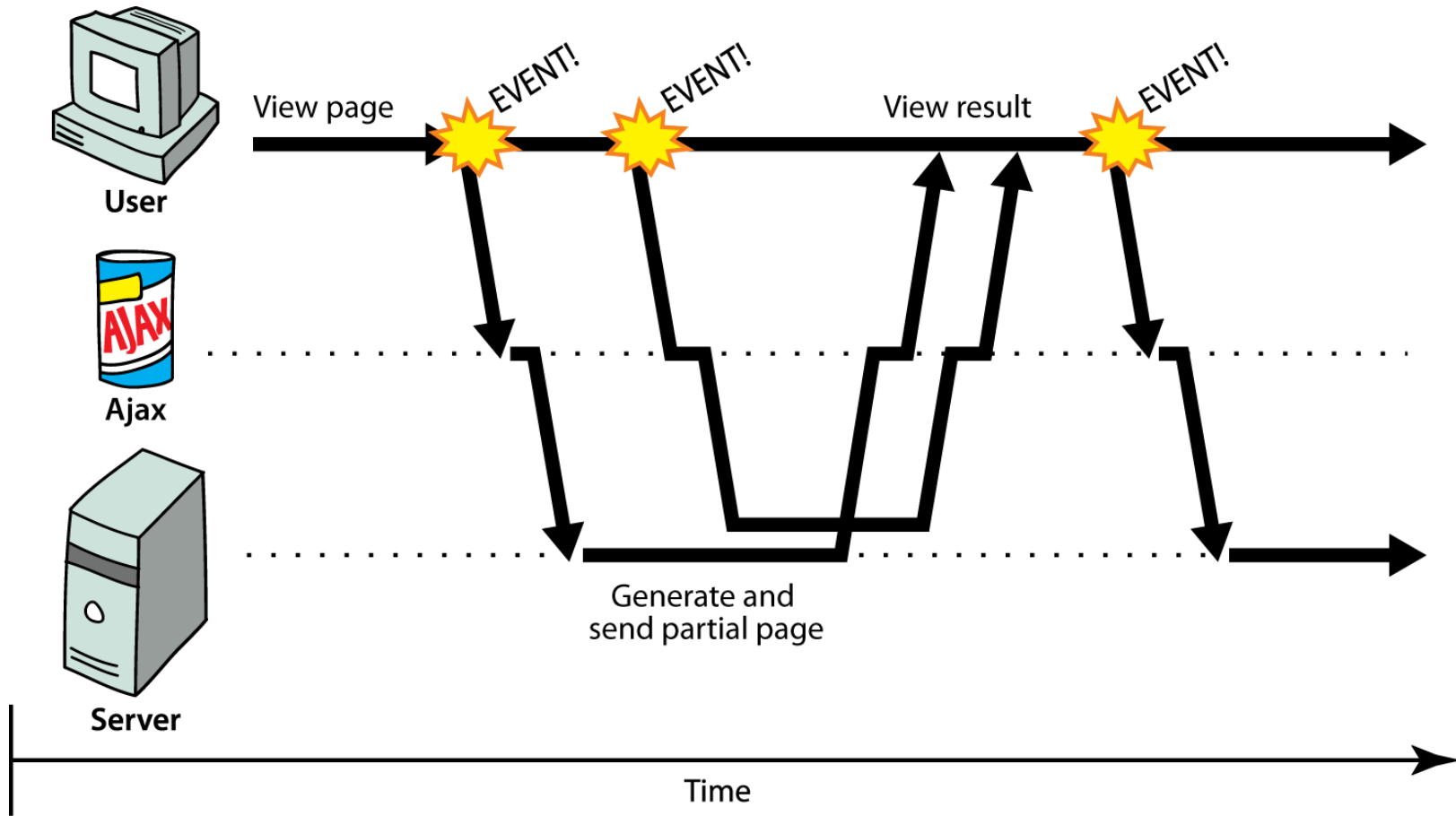
Асинхронное взаимодействие

- Сравним синхронное и асинхронное взаимодействия.
- Изучим Java Message Service (JMS).
- Узнаем что такое распределённые транзакции.
- Поговорим о протоколах обмена сообщениями.
- Вспомним корпоративную сервисную шину (ESB).

СИНХРОННОЕ ВЗАИМОДЕЙСТВИЕ



АСИНХРОННОЕ ВЗАИМОДЕЙСТВИЕ



Синхронное взаимодействие.

Преимущества:

- легко запрограммировать;
- результат сразу же;
- обычно механизм восстановления в случае ошибки проще;
- обычно более быстрый ответ.

Недостатки:

- сервис, с которым осуществляется взаимодействие, должен быть запущен и доступен;
- вызывающая сторона блокируется.

Асинхронное взаимодействие.

Преимущества:

- необязательно связывать запрос с конкретным сервером;
- необязательно сервис, с которым осуществляется взаимодействие, должен быть доступен;
- неблокирующий.

Недостатки:

- непредсказуемое время ответа;
- обычно механизм восстановления в случае ошибки сложнее;
- дизайн приложения сложнее.

Представляет результат асинхронного вычисления.

```
Future<Long> future = executor.submit(new
    Callable<Long>() {
        public Long call() throws Exception {
            ...
            return ...;
        }
    });
```

// Doing something while value is being calculated

```
Long result = future.get();
```

JAVA MESSAGE SERVICE (JMS)

JMS API – Java API, которое позволяет приложениям создавать, отправлять, получать и читать сообщения.

Входит в состав Java EE.

Создаёт взаимодействие, которое:

- *Асинхронное*

Сообщения доставляются клиенту по мере их поступления, клиенту не нужно делать дополнительные запросы на их получение.

- *Надёжное*

- Независимость компонента от интерфейсов других компонентов => компоненты можно легко заменять.
- Независимость работы приложения от того, что все компоненты стартованы и работают одновременно.
- Нет необходимости компоненту получить ответ от другого компонента сразу после отправки информации.

- JMS брокер

Система, которая имплементирует JMS интерфейсы и представляет средства для контроля и управления.

- JMS клиенты

- Сообщения

- Администрируемые объекты

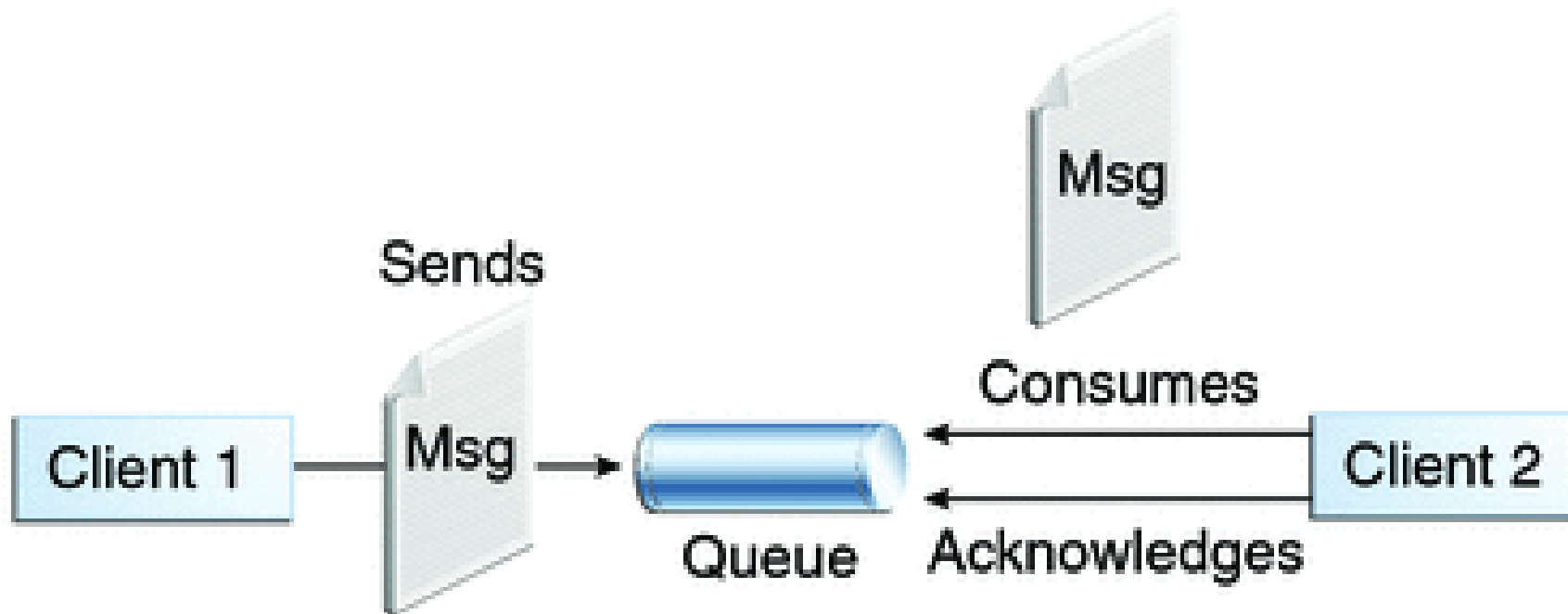
- JMS поставщик
- JMS клиенты
Компоненты, которые производят или потребляют сообщения.
- Сообщения
- Администрируемые объекты

- JMS поставщик
- JMS клиенты
- Сообщения
Объекты, которыми общаются JMS клиенты.
- Администрируемые объекты

- JMS поставщик
- JMS клиенты
- Сообщения
- Администрируемые объекты
Предконфигурированные JMS объекты, созданные администратором для использования клиентами: пункты назначения (destinations) и фабрики соединений (connection factories).

ТОЧКА-К-ТОЧКЕ (РТР)

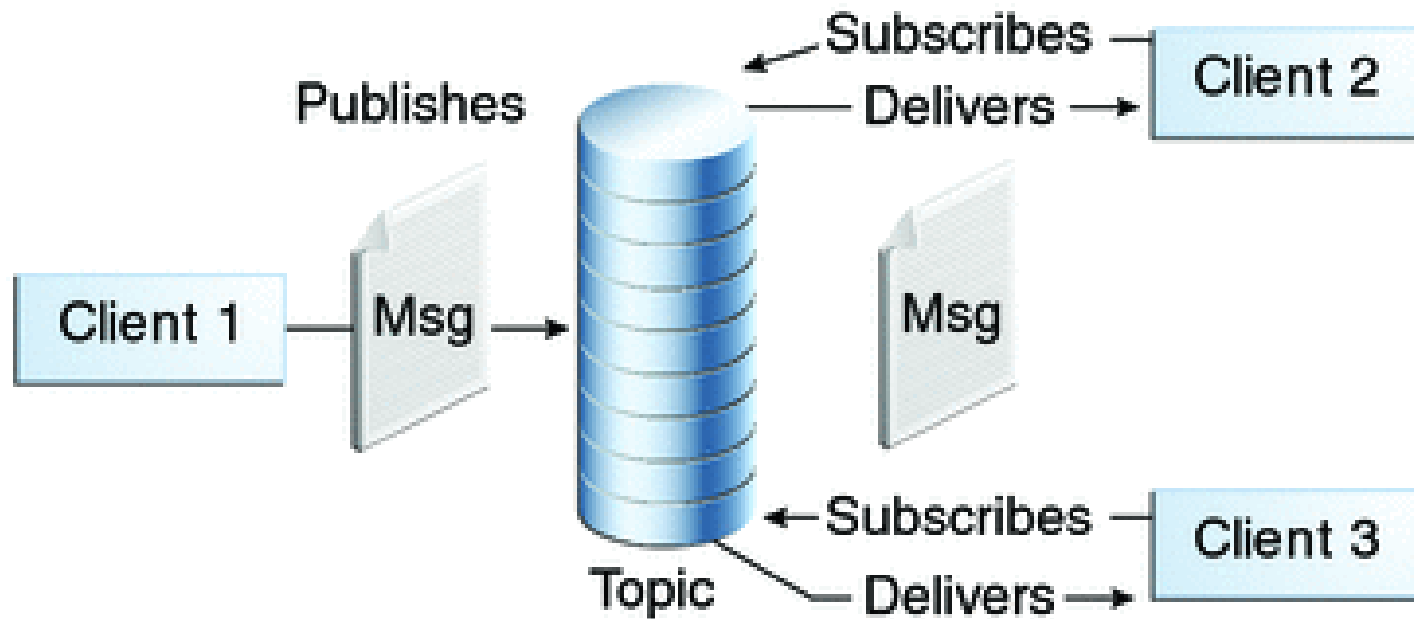
Концепция *очереди* сообщений.



- Каждое сообщение имеет только 1 потребителя.
- Получатель должен подтвердить успешную обработку сообщения.
- Если нет ни одного доступного потребителя в момент отправки сообщения, то оно будет сохранено до момента появления потребителя, который сможет его обработать.

ПУБЛИКАЦИЯ/ПОДПИСКА (PUB/SUB)

Концепция *тем*, на которые подписываются клиенты.



- Каждое сообщение может иметь множество потребителей.
- Подписчик темы получит только те сообщения, что были опубликованы после создания им подписки. Он должен оставаться активным, чтобы получать новые сообщения

- Синхронная.

Подписчик или получатель явно получают сообщения путём вызова метода `receive`, который блокируется до момента поступления сообщения.

- Асинхронная.

- Синхронная.

- Асинхронная.

Клиент может зарегистрировать слушателя сообщений (аналогично слушателю событий). В случае поступления сообщения провайдер вызовет метод `onMessage`.

- `javax.jms.ConnectionFactory`
- `javax.jms.Connection`
- `javax.jms.Session`
- `javax.jms.MessageProducer`
- `javax.jms.MessageConsumer`
- `javax.jms.Message`
- `javax.jms.Topic`
- `javax.jms.Queue`
- `javax.jms.MessageListener`
- `javax.jms.JMSException`

ТИПЫ JMS СООБЩЕНИЙ

Тип	Содержимое тела сообщения
TextMessage	Строка (java.lang.String)
MapMessage	Карта значений со строкой в качестве ключа
BytesMessage	Поток неинтерпретированных байтов
StreamMessage	Поток Java примитивов
ObjectMessage	Serializable объект
Message	Содержимого нет, только свойства и заголовки

А что, если нужно остановить читателей|подписчиков?

Подсуньте читателю|подписчику таблетку с ядом 😊

POISON PILL – сообщение с заранее известным содержанием. Читатели|подписчики должны ожидать такое сообщение и при получении его выполнить определённые действия.

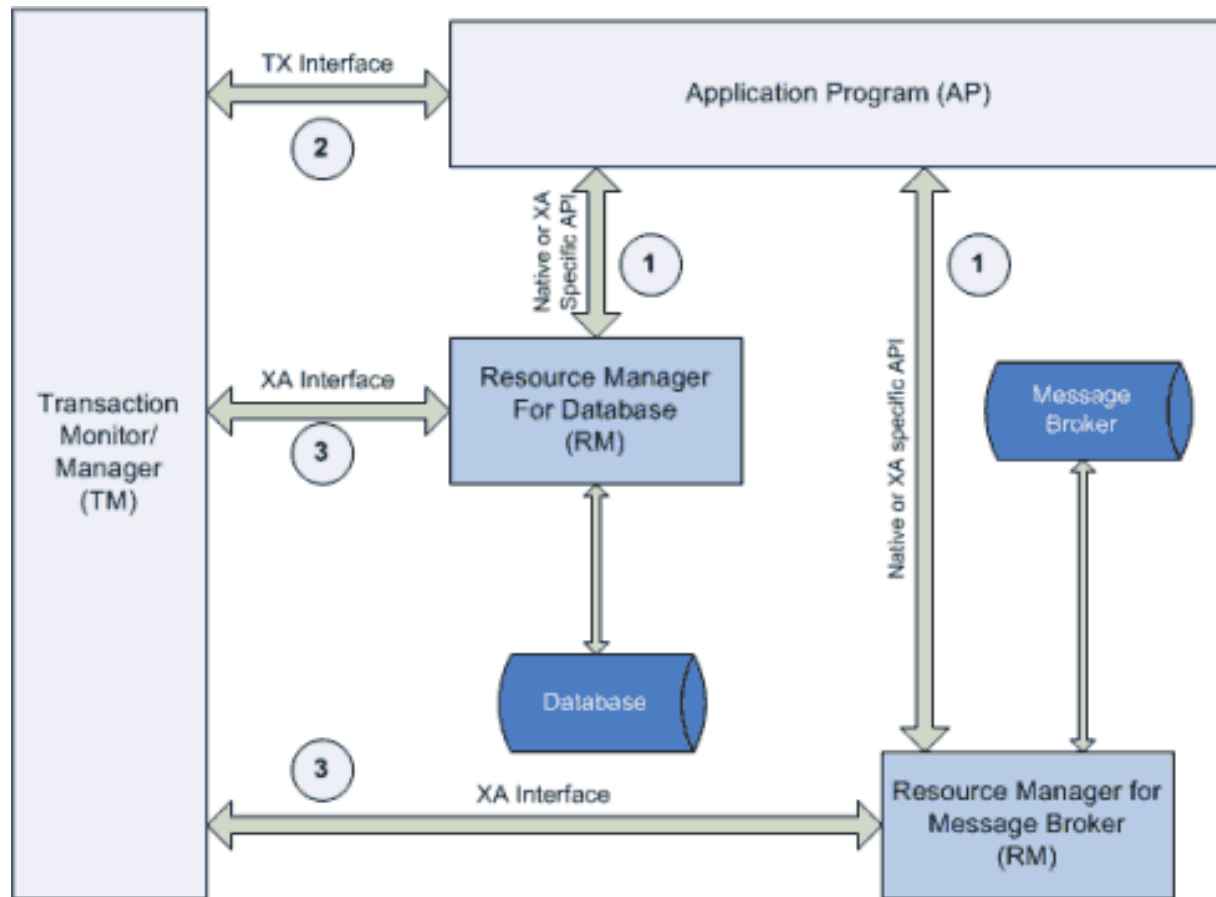
А что, если нужно фильтровать получаемые сообщения по определённым параметрам, т.е. получать только интересующие?

Используйте message selector – стандартный механизм JMS API. Он позволит фильтровать сообщения JMS провайдером, а не самим приложением.

Возможны следующие варианты:

- автоматический acknowledgement;
- явный acknowledgement (`Message.acknowledge()`);
- JMS транзакции;
- распределённые (XA) транзакции.

РАСПРЕДЕЛЁННЫЕ (XA) ТРАНЗАКЦИИ



- Mockito.
Не забывайте, что можно мокать интерфейсы (в том числе и JMS API).
- Embedded JMS providers.
 - Не нужно запускать отдельный процесс брокера.
 - Можно отключать persistence, что необходимо для повторяемости и изолированности тестов.
 - Простота конфигурации (например, с помощью Spring).

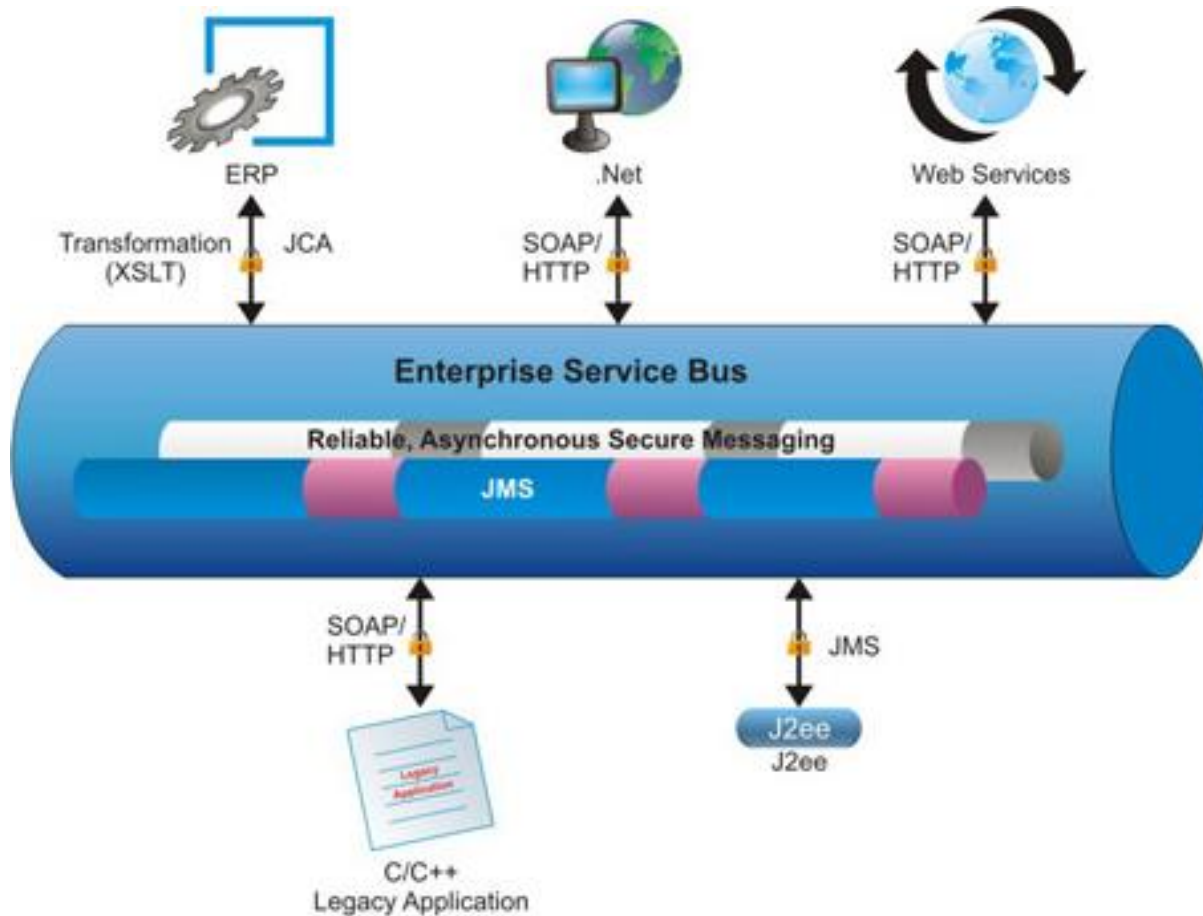
Существуют различные *протоколы* обмена между клиентом и брокером сообщений. Они специфицируют каждый байт передаваемых данных => исключают ограничение на язык программирования.

- AMQP – Advanced Message Queuing Protocol.
- MQTT – machine-to-machine (M2M)/"Internet of Things" connectivity protocol.
- STOMP – Simple (or Streaming) Text Orientated Messaging Protocol.
- ...

- ActiveMQ
Поддерживает JMS API, AMQP, STOMP, MQPP.
- RabbitMQ
Поддерживает AMQP, STOMP, MQPP, HTTP.
- Hazelcast
In-memory data grid (IMDG) с поддержкой брокера сообщений.
- Kafka
Распределённая потоковая платформа.

Сервисная шина предприятия (ESB) — модель архитектуры ПО, используемая для проектирования и реализации связи между взаимно взаимодействующими приложениями в сервисно-ориентированной архитектуре (SOA).

СЕРВИСНАЯ ШИНА (ESB)



- <https://docs.oracle.com/cd/E19798-01/821-1841/bncdq/index.html>
- <http://activemq.apache.org/faq.html>
- <https://www.rabbitmq.com/getstarted.html>
- <http://www.javaworld.com/article/2077714/java-web-development/xa-transactions-using-spring.html>