RabbitMQ消息中间件

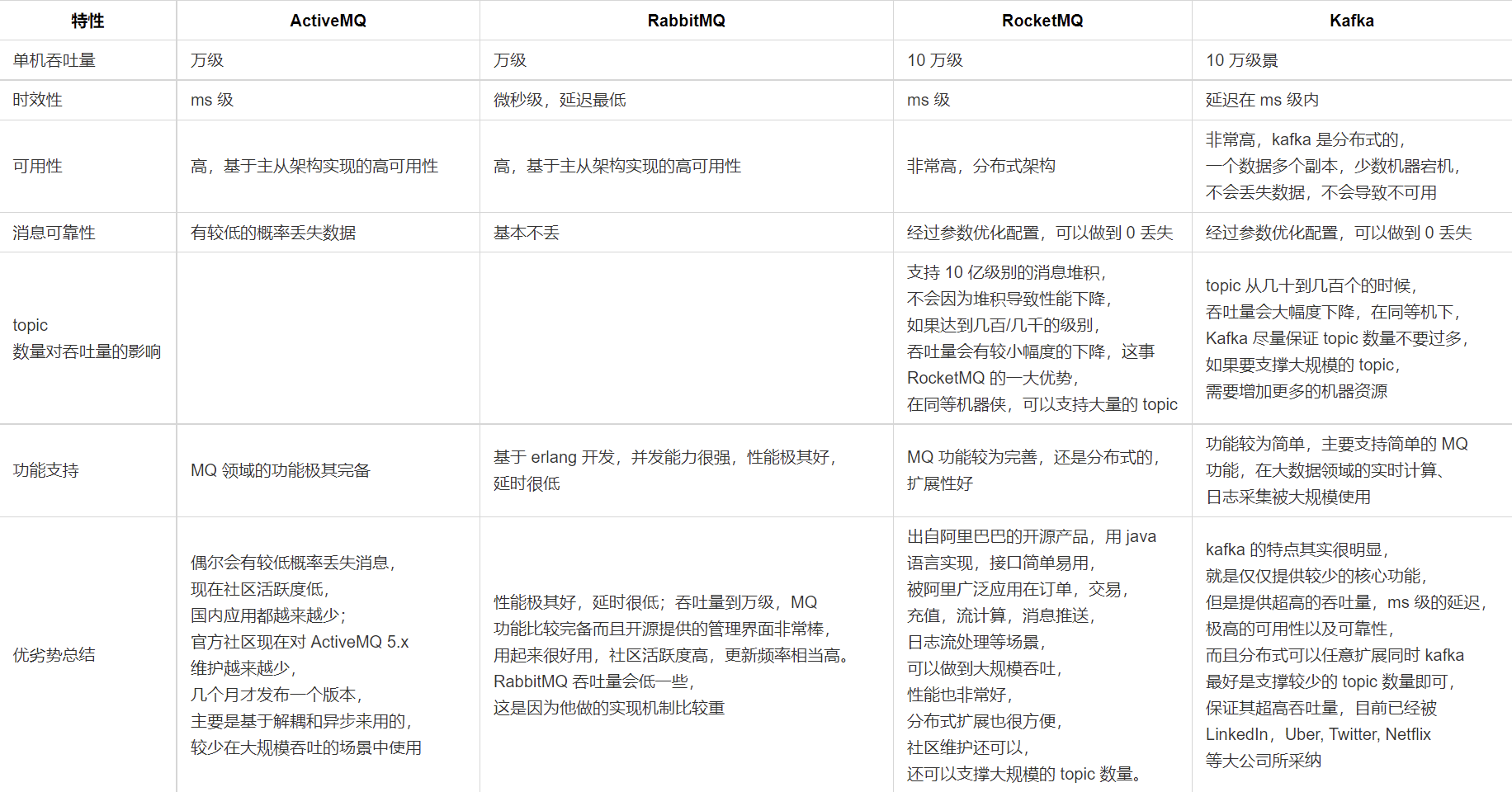
# **基本概念**

MQ (message Queue) 消息队列，是将消息 message 存放到队列中（队列：先进先出 FIFO 的数据结构 ），它是一种跨进程的通信机制，用于上下游的消息传递，从而达到上下游解耦，也就是说消息发送上游只需要依赖 MQ，不需要依赖其他下游服务。

在系统架构设计中，经常会使用 MQ，常见的使用场景有异步处理、应用解耦、流量削峰、消息通讯、日志处理等；但是使用消息队列，也有弊端，比如增加了复杂度，存在一致性问题等。

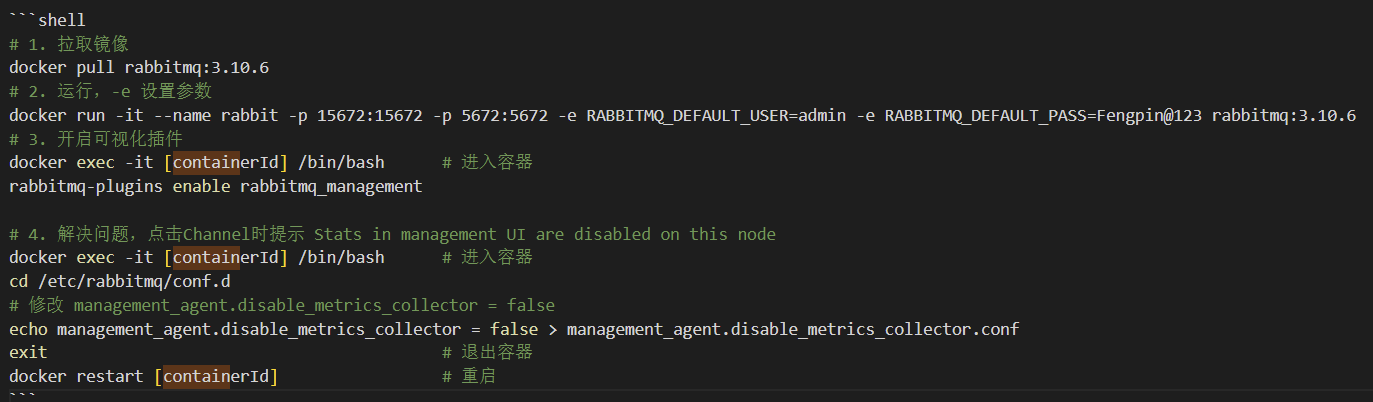
常用的消息中间件有ActiveMQ、RabbitMQ、RocketMQ、Kafka等。

# **技术选型**



RabbitMQ 实现了高级消息队列协议（AMQP）的开源消息中间件。它通过直接支持或插件支持的方式，支持了几种消息协议：AMQP、XMPP、SMTP、STOMP，也正因如此，它非常重量级，更适合于企业级的开发。它是用Erlang语言编写的，结合 erlang 语言本身的并发优势，性能好，时效性微秒级，社区活跃度也比较高，管理界面用起来十分方便，如果数据量没有那么大，中小型公司优先选择功能比较完备的 RabbitMQ。

# **Docker安装**



# **核心组件**

1. Connection：代表 AMQP 0-9-1 连接；通过 ConnectionFactory 构建 Connection 实例，Connection（连接）用于开启通道；
2. Channel：代表 AMQP 0-9-1 通道，提供了大多数操作协议方法，承载生成者、消费者、队列的交互；
3. Exchange：交换机，接受生产者生产的消息；
4. Queue：队列；
5. Binding：绑定交换机和队列；
6. RoutingKey：路由键，每一条消息的具体绑定关系表达式；

# **消息模型**

RabbitMQ 提供了 7 种消息模型。第 6 种是 RPC 拉取方式，基本不用；第 7 种是新出的发布者确认模式，也基本不用；3、4、5 这三种都属于订阅模型，只不过进行路由的方式不同。

## **Simple模型**

最简单的消息模式，使用的是默认交换机。

## **Work 模型**

多个消费者绑定到一个队列，共同消费队列中的消息；队列中的消息一旦消费，就会消失，每个消费者获取到的消息唯一，使用的也是默认交换机。

## **Fanout 模型**

交换机和列队直接绑定，不需要指定 routingkey，所以他的消息传输速度是发布订阅模式中最快的。

## **Direct 模型**

与 fanout 模式相比，direct 模式需要 RoutingKey 将队列和交换机绑定。

## **Topic 模型**

与之前的两种模式相比，区别在于：它可以通过 RoutingKey，将交换机和队列机进行模糊匹配绑定，满足较复杂的生产消息存放到队列的匹配过程。

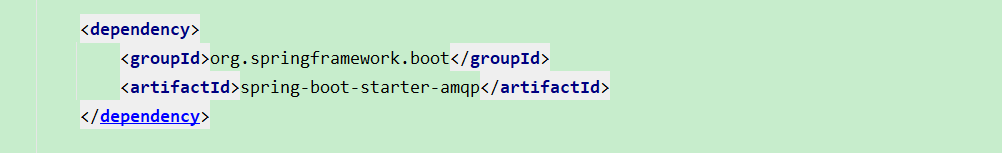
\*，有且只匹配一个词

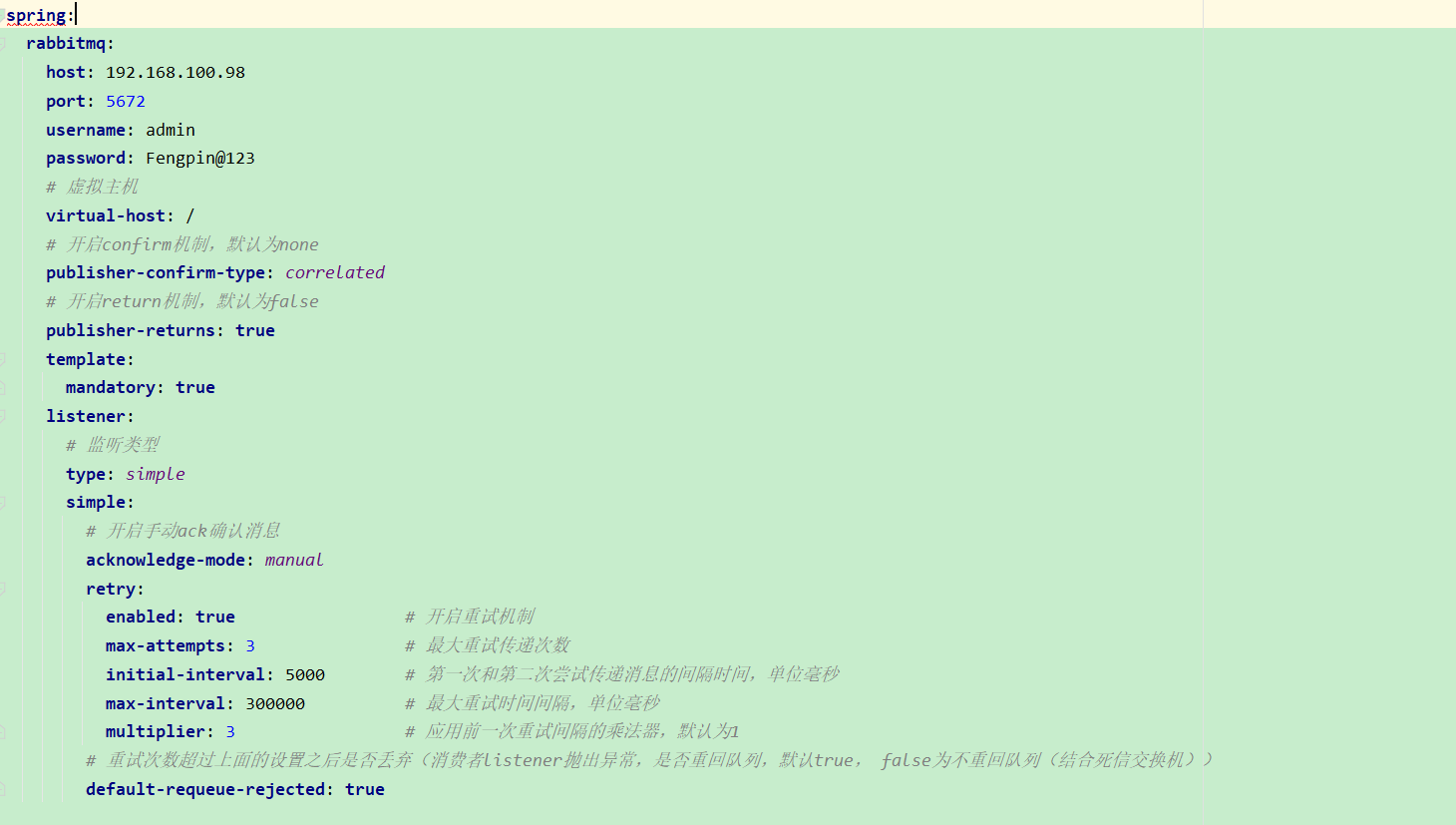
#，匹配一个或多个词

# **Demo实例**

## **生产者代码**

1. 导入jar包，编写yaml配置





1. 增加配置类；





1. 增加消息发送异常回退处理器；



1. 发送消息通用业务类；



1. 业务代码中直接调用。

## **消费者代码**

1. 导入依赖，编写yaml配置，同上；
2. 增加配置类，同上；
3. 增加消费者接受消息逻辑；



1. 获取到数据后调用业务方法处理。

# **常见问题说明**

## **交换机、队列、绑定关系谁来维护？**

理想情况下,生产者应该只知道exchange(而不是队列),并且消费者应该只知道队列(不是交换器)，并将队列绑定到交换机；如果消费者增加新队列，生产者则不需要修改，但是有个问题，消费者部署之前，生产者启动后发出的所有消息都会丢失。

解决方案：在生成者和消费者都维护。

## **消息丢失问题**

在实际项目中，为了保证消息的可靠性，生产者都会开启消息确认机制（confirm、return）,MQ中的消息持久化，消费者手动ACK应答等措施。