### Rabbit Mq在业务中的问题及解决方案：

1. mq在业务中解决的问题

mq在本次业务中，通过异步传输消息的方式，解决了业务中请求过量而影响整体服务器的运行效果问题。（在Contorller层接收请求并发送消息至队列中，等待消费端的消费）

1. mq消费端失败后的解决方案

在消息发送至队列后，消费端开始进行消费，但由于消费端运行业务时产生了预计之外 的错误使得消息消费失败，此时mq自身有消息的重试功能，默认的消息重试是无限 的，这并不符合业务的需求。

1. 失败后重试次数、时长、是否丢弃失败消息、手自动控制等。

利用springboot的自动配置，在配置文件中设置了：是否开启消费者重试、

最大重试次数、重试时间间隔、消息是否丢弃、ack返回手自动控制等。

1. 最终失败的消息发送至死信队列的记录

如果失败的消息重试后最终依然失败，采取死信队列的方式。在创建业务队列的同时创建死信队列并且进行绑定，最终失败消息会发送至死信队列并进行日志记录或者储存至数据库内设置定时任务进行后续重试。

1. 失败消息在重试消费过程中总是优先在队列第一位，对其进行的优化 (预留问题)

失败消息在设置最大限制内的进行重试时，失败的信息总是优先排列在队列的最前端，不符合消息的排列顺序和业务的理想状态。

问题1: 如何将失败消息后置到队列后方，进行有序的排列。

问题2: 失败消息与重试时间的控制。

问题3: 后续如何判断消息的重试次数。

mq的运行机制：

1. 首先生产者生产消息发送至业务交换机-(business-exchange)

2. 业务交换机根据Key-(business\_queue\_key)匹配相应的业务队列-(business-queue)

3. 消息通过队列发送给消费者进行消费。

4. 在消费过程中出现消费失败的情况导致消费端重试，设置的次数为3次，间隔3秒。

5. 消费者重试消费3次后依然失败，预把消息发送至死信队列中。

6. 首先消息通过业务队列中绑定的死信交换机- (dead-letter-exchange)和

Key- (dead-letter-key)匹配相应的死信队列- (dead-letter-queue)，

消息被发送至死信队列中。

1. 消息最终在死信队列中进行日志记录或者储存至数据库内设置定时任务进行后续 重试。

springboot融合mq的自动配置：

基本连接配置：

spring.rabbitmq.host=127.0.0.1  
 spring.rabbitmq.port=5672  
 spring.rabbitmq.username=admin  
 spring.rabbitmq.password=admin  
 spring.rabbitmq.publisher-confirms=true  
 spring.rabbitmq.virtual-host=/

升级配置：  
 spring.rabbitmq.listener.simple.retry.max-attempts=3 //最大重试次数  
 spring.rabbitmq.listener.simple.retry.enabled=true //是否开启消费端重试  
 spring.rabbitmq.listener.simple.retry.initial-interval=3000 //重试时间间隔  
 spring.rabbitmq.listener.simple.acknowledge-mode=auto //ack机制的自动控制  
 spring.rabbitmq.listener.simple.default-requeue-rejected=false //最终消费失败 信息的是否丢弃

等... ...