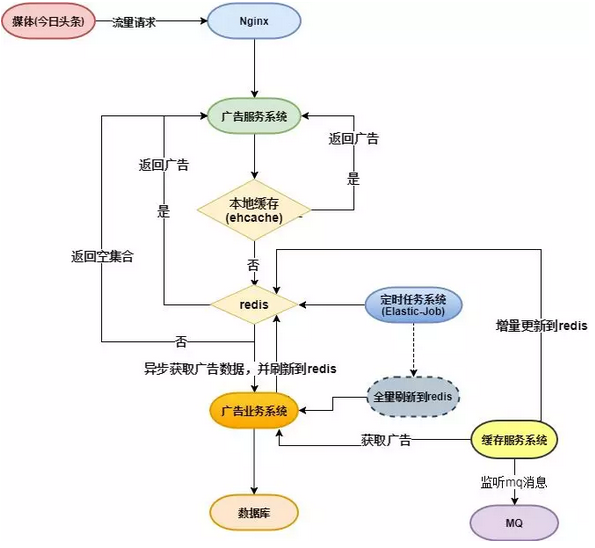
**一、背景介绍：消息可靠传递的重要性**

比如：某个广告主（如：天猫）想在我们的平台（如：今日头条）投放广告，当通过我们的广告系统新建广告的时候，该消息在同步给redis缓存(es)的时候丢失了，而我们又没有发现，造成该广告无法正常显示出来，那这损失就大了，如果1天都没有该广告的投放记录，那就有可能是上百万的损失了，所以消息的可靠传输对我们的广告系统也是很重要的。 其实，生活中这样的场景很常见，再比如：交易系统、订单系统都必须保证消息的可靠传输，否则，损失是巨大的！！！



**二、如何保证消息的可靠传递呢？**

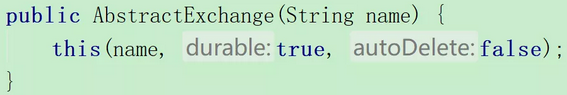
**1、设置交换机、队列和消息都为持久化**

\*\*持久化：\*\*保证在服务器重启的时候可以保持不丢失相关信息，重点解决服务器的异常崩溃而导致的消息丢失问题。但是，将所有的消息都设置为持久化，会严重影响RabbitMQ的性能，写入硬盘的速度比写入内存的速度慢的不只一点点。对于可靠性不是那么高的消息可以不采用持久化处理以提高整体的吞吐率，在选择是否要将消息持久化时，需要在可靠性和吞吐量之间做一个权衡。 处于某种应用场景，如：大流量的订单交易系统，为了不影响性能，我们可以不设置持久化，但是我们会定时扫描数据库中的未发送成功的消息，进行重试发送，实际应用场景，我们其实有很多解决方案，不要故步自封，换个角度多想想，只有经历多了，才能应用的更加得心应手。

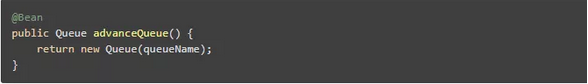
1）交换机的持久化



注释：查看源码，易知，默认是持久化的



2）队列的持久化



注释：查看源码，易知，默认是持久化的

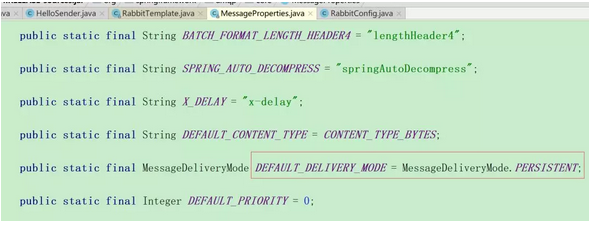


3）消息的持久化

当我们使用RabbitTemplate调用了 convertAndSend(String exchange, String routingKey, final Object object) 方法。默认就是持久化模式

注意：

* 持久化的消息在到达队列时就被写入到磁盘，并且如果可以，持久化的消息也会在内存中保存一份备份，这样可以提高一定的性能，只有在内存吃紧的时候才会从内存中清除。
* 非持久化的消息一般只保存在内存中，在内存吃紧的时候会被换入到磁盘中，以节省内存空间。



**2、生产者消息确认机制**

当消息发送出去之后，我们如何知道消息有没有正确到达exchange呢？如果在这个过程中，消息丢失了，我们根本不知道发生了什么，也不知道是什么原因导致消息发送失败了 为解决这个问题，主要有如下两种方案：

* 通过事务机制实现
* 通过生产者消息确认机制（publisher confirm）实现

但是使用事务机制实现会严重降低RabbitMQ的消息吞吐量，我们采用一种轻量级的方案——生产者消息确认机制

什么是消息确认机制？ 简而言之，就是：生产者发送的消息一旦被投递到所有匹配的队列之后，就会发送一个确认消息给生产者，这就使得生产者知晓消息已经正确到达了目的地。 如果消息和队列是持久化存储的，那么确认消息会在消息写入磁盘之后发出。 再补充一个Mandatory参数：当Mandatory参数设为true时，如果目的不可达，会发送消息给生产者，生产者通过一个回调函数来获取该信息。

**3、消费者消息确认机制**

为了保证消息从队列可靠地到达消费者，RabbitMQ提供了消费者消息确认机制（message acknowledgement）。采用消息确认机制之后，消费者就有足够的时间来处理消息，不用担心处理消息过程中消费者进程挂掉后消息丢失的问题，因为RabbitMQ会一直等待并持有消息，直到消费者确认了该消息。

**4、死信队列**

DLX，Dead Letter Exchange 的缩写，又死信邮箱、死信交换机。DLX就是一个普通的交换机，和一般的交换机没有任何区别。 当消息在一个队列中变成死信（dead message）时，通过这个交换机将死信发送到死信队列中（指定好相关参数，rabbitmq会自动发送）。

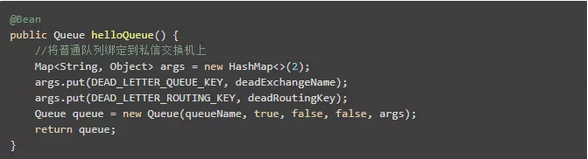
什么是死信呢？什么样的消息会变成死信呢？

* 消息被拒绝（basic.reject或basic.nack）并且requeue=false.
* 消息TTL过期
* 队列达到最大长度（队列满了，无法再添加数据到mq中）

应用场景分析： 在定义业务队列的时候，可以考虑指定一个死信交换机，并绑定一个死信队列，当消息变成死信时，该消息就会被发送到该死信队列上，这样就方便我们查看消息失败的原因了 \*\*如何使用死信交换机呢？

定义业务（普通）队列的时候指定参数：

* x-dead-letter-exchange: 用来设置死信后发送的交换机
* x-dead-letter-routing-key：用来设置死信的routingKey



**三、实战演练**

项目代码下载地址：https://gitee.com/jikeh/JiKeHCN-RELEASE.git 项目名：spring-boot-rabbitmq-reliability

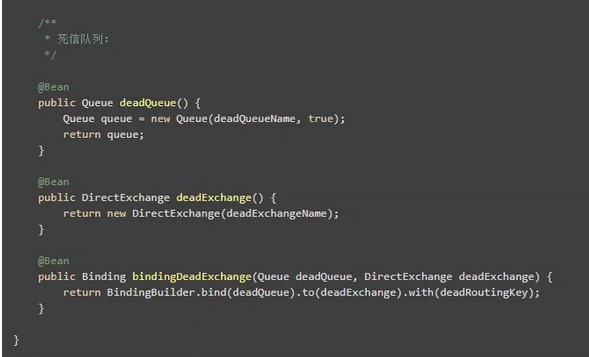
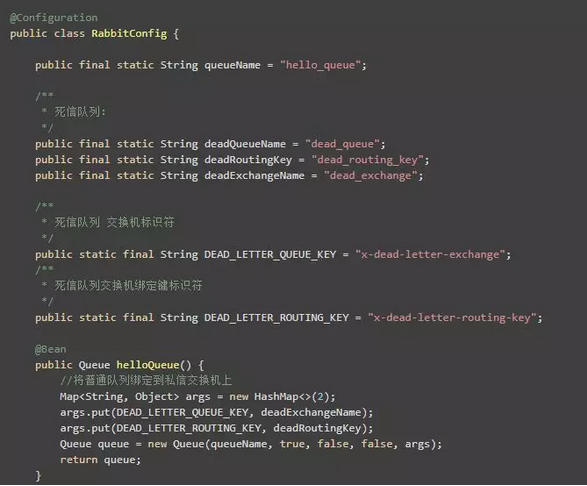
**1、开启生产者消息确认机制**



**2、开启消费者消息确认机制**



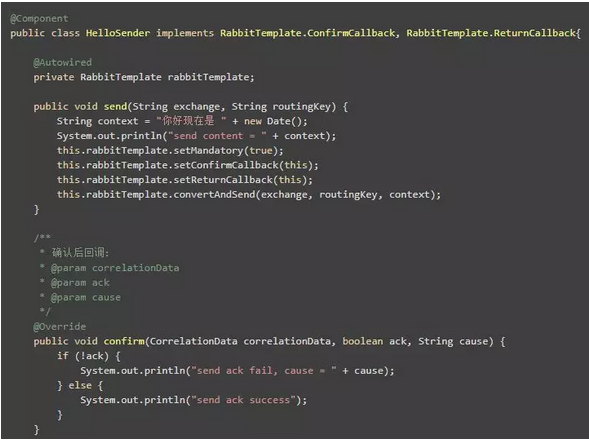
**3、基本配置**



注释：hell\_queue就配置了死信交换机、死信队列



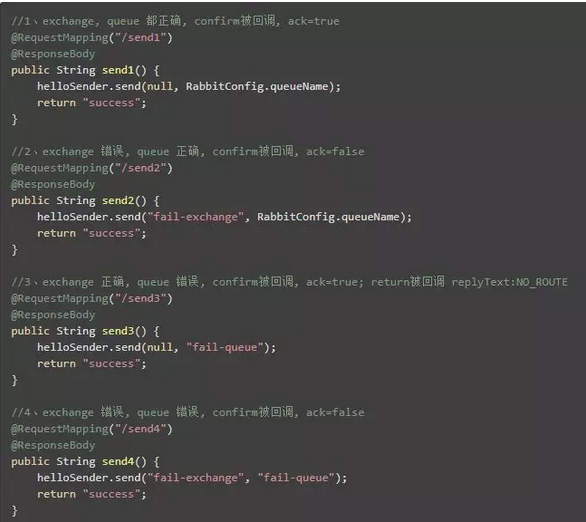
**4、生产者核心代码**



**5、消费者核心代码**



**6、测试生产者消息确认功能：分为4种场景来测试**

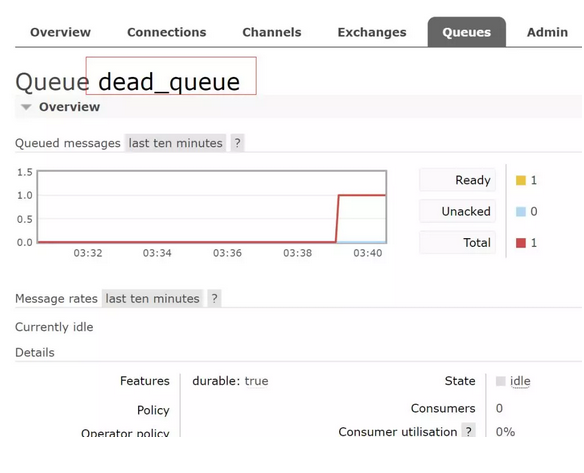


**7、测试消费者消息确认功能**

* 1）当添加这行代码的时候： channel.basicAck(message.getMessageProperties().getDeliveryTag(),false); 测试结果：消息被正常消费，消息从队列中删除
* 2）当注释掉这行代码的时候： channel.basicAck(message.getMessageProperties().getDeliveryTag(),false); 测试结果：消息会被重复消费，一直保留在队列当中

**8、测试死信队列**

当执行这行代码的时候： channel.basicNack(message.getMessageProperties().getDeliveryTag(), false,false); 消息会被加入到死信队列中：



**四、拓展**

除了我们上面讲的基本可靠性保证外，其实还有很多性能优化方案、可靠性保证方案：集群监控、流控、镜像队列、HAProxy+Keeplived高可靠负载均衡 我们后续会继续分享上述内容，欢迎持续关注…… 下节课，我们将会将该功能应用到缓存架构上了