deliveryTag（唯一标识 ID）：当一个消费者向 RabbitMQ 注册后，会建立起一个 Channel ，RabbitMQ 会用 basic.deliver 方法向消费者推送消息，这个方法携带了一个 delivery tag， 它代表了 RabbitMQ 向该 Channel 投递的这条消息的唯一标识 ID，是一个单调递增的正整数，delivery tag 的范围仅限于 Channel，delivery\_tag是消息投递序号，每个channel对应一个(long类型)，从1开始到9223372036854775807范围，在手动消息确认时可以对指定delivery\_tag的消息进行ack、nack、reject等操作。

每次消费或者重新投递requeue后，delivery\_tag都会增加，理论上该正常业务范围内，该值永远不会达到最大范围上限。可以根据每个消费者对应channel的delivery\_tag消费速率计算到达最大值需要的时间。

假设：每秒钟一个消费者可以消费1000w个消息(假设每个消费者一个channel)，则 9223372036854775807 / (60 \* 60 \* 24 \* 365 \* 1000w) = 29247年后能达到上限数值。

CachingConnectionFactory  
CachingConnectionFactory为我们提供了两种缓存的模式：

* CHANNEL模式：这也是CachingConnectionFactory的默认模式，在这种模式下，所有的createConnection（）方法实际上返回的都是同一个Connection，同样的Connection.close()方法是没用的，因为就一个，默认情况下，Connection中只缓存了一个Channel，在并发量不大的时候这种模式是完全够用的，当并发量较高的时候，我们可以setChannelCacheSize（）来增加Connection中缓存的Channel的数量。
* CONNECTION模式：在CONNECTION模式下，每一次调用createConnection（）方法都会新建一个或者从缓存中获取，根据你设置的ConnectionCacheSize的大小，当小于的时候会采用新建的策略，当大于等于的时候会采用从缓存中获取的策略，与CHANNEL模式不同的是，CONNECTION模式对Connection和Channel都进行了缓存，最新版本的client中已经将Channel的缓存数量从1增加到了25，但是在并发量不是特别大的情况下，作用并不是特别明显。  
  **使用CachingConnectionFactory需要注意的一点是：所有你获取的Channel对象必须要显式的关闭，所以finally中一定不要忘记释放资源，如果忘记释放，则可能造成连接池中没有资源可用**。

rabbitmqTemplate的convertAndSend与send方法底层都是调用rabbitmqTemplate的send方法，区别为参数列表不同，第一种可以加correlationData，用于可靠性投递，表明消息的唯一性标志，而send方法无此参数

**Jackson2JsonMessageConverter**

MessageProperties messageProperties = new MessageProperties();

messageProperties.setContentType("application/json");

设置contentType类型，消费端将消息转换为Map类型，而非字节数组，当传输的为实体的list集合时，接收端接收到的参数为List<T>

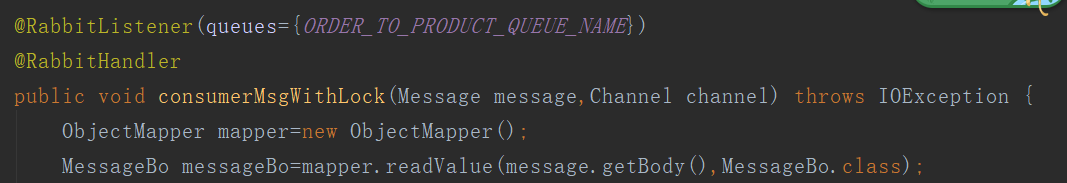




将发送的消息的对象转换为对象实体，对于跨系统的情况，发送消息的类和接受消息的类必须是一样的，不仅是要里面的字段一样，类名一样，连类的包路径都要一样。

所以当系统1使用 JsonMessageConverter 发送消息类A给系统2时，系统2可以有如下几种方式来接收：

* 1.依赖系统1的jar包,直接使用类A来接收
* 2.不依赖系统1的jar包，自己建一个和A一模一样的类，连名称，包路径都一样
* 3.负责监听 queue 的类实现 MessageListener 接口，直接接收 Message 类，再自己转换



项目中一般使用第三种情况，如果第二种情况（通过配置SimpleMessageListenerContainer实现消息的监听）也可以通过在消息的生产端设置消费端消息对应实体的全限定名，或者在消费端解析消息之前修改TypeId为消费端消息对应实体的全限定名

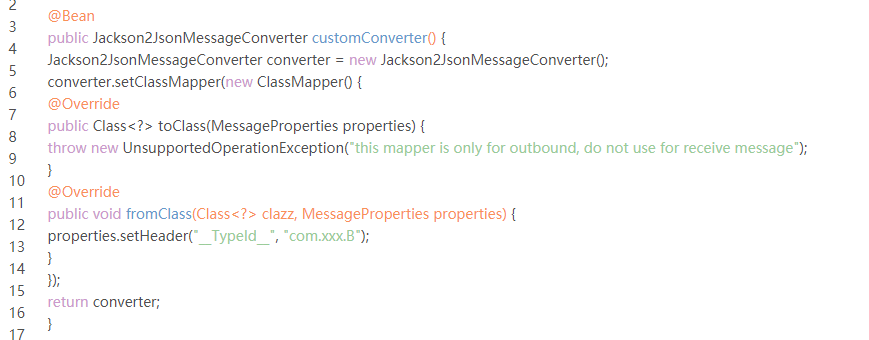
MessageProperties messageProperties = new MessageProperties();

messageProperties.setContentType("application/json");

//指定的\_\_TypeId\_\_属性值必须是消费端的Order的全类名，如果不匹配则会报错。

messageProperties.getHeaders().put("\_\_TypeId\_\_","com.zhihao.miao.test.day10.Sender.Order");

在消息的生产者端为 JsonMessageConverter， 设置一个自定义的 ClassMapper，重写 fromClass 方法，将 \_\_TypeId\_\_ 的值设为消费端用来接收的类的路径+名称。当然了，也可以在消费者端重写toClass方法，直接返回想要转换的目标类的类类型。两种选一种就可以



使用SimpleMessageListenerContainer的一个优点是可以根据消息传输的集体类型走具体的逻辑

运用方法的重载

