基于MQ消息的数据最终一致性方案和规范

版本 <V1.0>

|  |  |
| --- | --- |
| 项目名称 | 版本： <V1.0> |
| 软件架构设计文档 | 日期： <dd/mmm/yy> |
| <sad> | |

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| <日/月/年> | <x.x> | <详细信息> | <姓名> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 范围 4

1.3 定义、首字母缩写词和缩略语 4

1.4 参考资料 4

1.5 说明 5

2. 整体方案说明 5

2.1 简介 5

2.2 构架设计面临的问题 5

2.3 构架目标和解决方案 5

2.4 消费方消息幂等性 12

3. 解决方案使用用例 12

3.1 消息发送方用例 13

3.2 消息接收方用例 14

3.3 核心配置用例 15

4. 逻辑时序图说明 16

4.1 消息发送 16

4.2 消息接收 16

4.3 架构模式 16

4.4 设计机制 16

5. 性能 17

6. 可扩展性 17

7. 其它说明 17

8. 附录A 指南 17

9. 附录B 规范 17

基于RabbitMQ消息的数据最终一致性方案

# 简介

架构在微服务基础上的多个子系统，不同于单一架构应用(Monolith), 分布式环境下, 进行事务操作将变得困难, 因为分布式环境通常会有多个数据源, 多个应用程序，很多业务场景下都会面临事务一致性方面的需求，只用本地数据库事务难以保证多个数据源数据的一致性。这种情况下, 可以使用两阶段或者三阶段提交协议来完成分布式事务。但是使用这种方式有如下缺点：

* 性能较差, 因为事务管理器需要在多个数据源之间进行多次等待
* 实现成本较高，需要使用XA或者TCC协议
* 两阶段或者三阶段提交需要每个实现接口有commit和rollback操作

有一种方法同样可以解决分布式事务问题, 并且性能较好, 这就是基于消息的数据最终一致性方案,本地事务以及消息队列来实现分布式事务。

## 目的

基于现有架构在微服务基础上的各个子系统，相互之间依赖调用，一个重要功能往往可能有跨系统多个接口调用来实现，数据的一致性，事务很难保证，并且可能系统出现bug，某几个接口调用失败，导致业务失败，数据不一致的问题，此方案就是为了解决此类问题制定的统一规范和解决方案。

## 范围

此软件构架文档适用的范围为龙翼系统中所有子模块。

## 定义、首字母缩写词和缩略语

软件构架文档所需的全部术语的定义、首字母缩写词和缩略语。

* ACID特性：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）。
* 分布式领域CAP理论：1.Consistency(一致性) 2.Availability(可用性) 3.Partition tolerance(分区容错性)
* 2PC (two-phase commit)：
* 3PC (three-phase commit)
* Paxos算法：莱斯利·兰伯特（英语：Leslie Lamport，LaTeX中的“La”）于1990年提出的一种基于消息传递且具有高度容错特性的一致性算法。基于消息传递通信模型的分布式系统，不可避免的会发生以下错误：进程可能会慢、被杀死或者重启，消息可能会延迟、丢失、重复，在基础Paxos场景中，先不考虑可能出现消息篡改即拜占庭错误的情况。Paxos算法解决的问题是在一个可能发生上述异常的分布式系统中如何就某个值达成一致，保证不论发生以上任何异常，都不会破坏决议的一致性。一个典型的场景是，在一个分布式数据库系统中，如果各节点的初始状态一致，每个节点都执行相同的操作序列，那么他们最后能得到一个一致的状态。为保证每个节点执行相同的命令序列，需要在每一条指令上执行一个“一致性算法”以保证每个节点看到的指令一致。一个通用的一致性算法可以应用在许多场景中，是分布式计算中的重要问题。因此从20世纪80年代起对于一致性算法的研究就没有停止过。

## 参考资料

RabbitMQ官网：

<https://www.rabbitmq.com/documentation.html>

Spring Framework官网：

<https://projects.spring.io/spring-framework/>

## 说明

定理：任何分布式系统只可同时满足二点，没法三者兼顾。

忠告：架构师不要将精力浪费在如何设计能满足三者的完美分布式系统，而是应该进行取舍。

# 整体方案说明

## 简介

基于现在公司采用的架构，使用RabbitMQ的消息事务机制，Spring-retry机制和RabbitMQ的x-dead-letter机制来实现基于消息的最终一致性性方案。

## 构架设计面临的问题

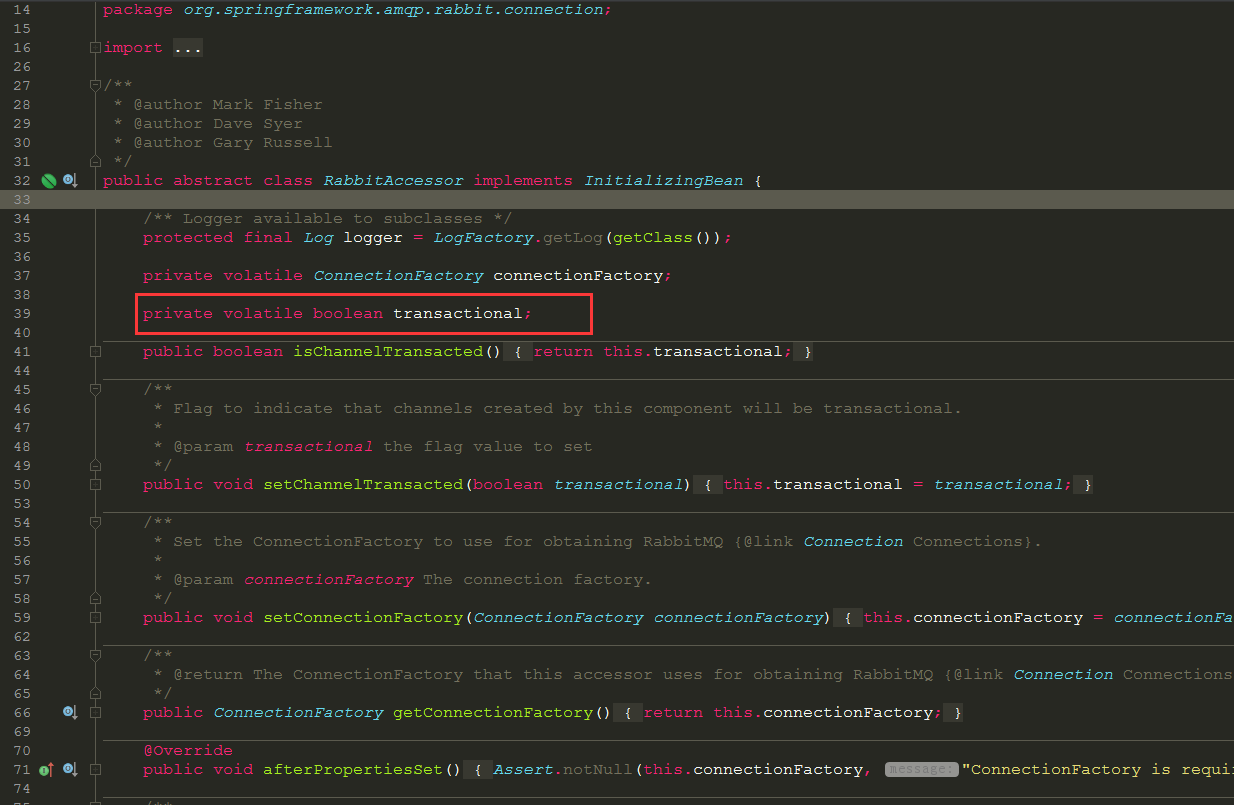
此方案要解决的核心问题有以下几点：

1. 如何让消息提供方本地数据库操作和发送消息到队列这两个操作成为一个原子操作，保持事务一致性
2. 如何让消费端接收消息和本地数据库操作这两个操作成为一个原子操作，保持事务一致性
3. 消费方如何保证接收到消息，一旦操作失败可以执行retry操作，策略和时间间隔可设置
4. 消费方一旦消息操作失败，到达重试次数后保证消息队列的非阻塞性，消息的不可丢失性
5. 如果消费方问题解决，如何将失败消息重新推送给它执行，以保证消费方数据的最终一致性

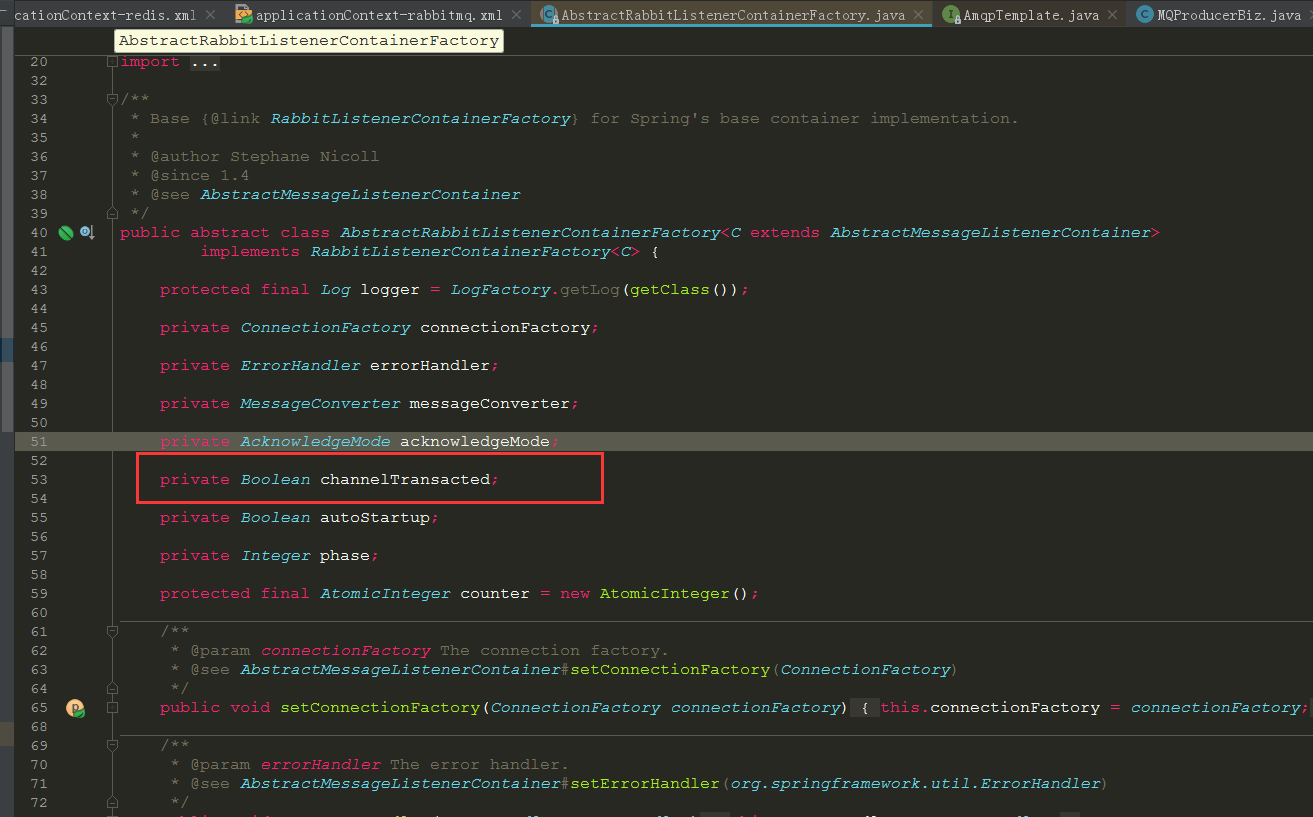
## 构架目标和解决方案

针对上一节的痛点问题，解决方案如下：

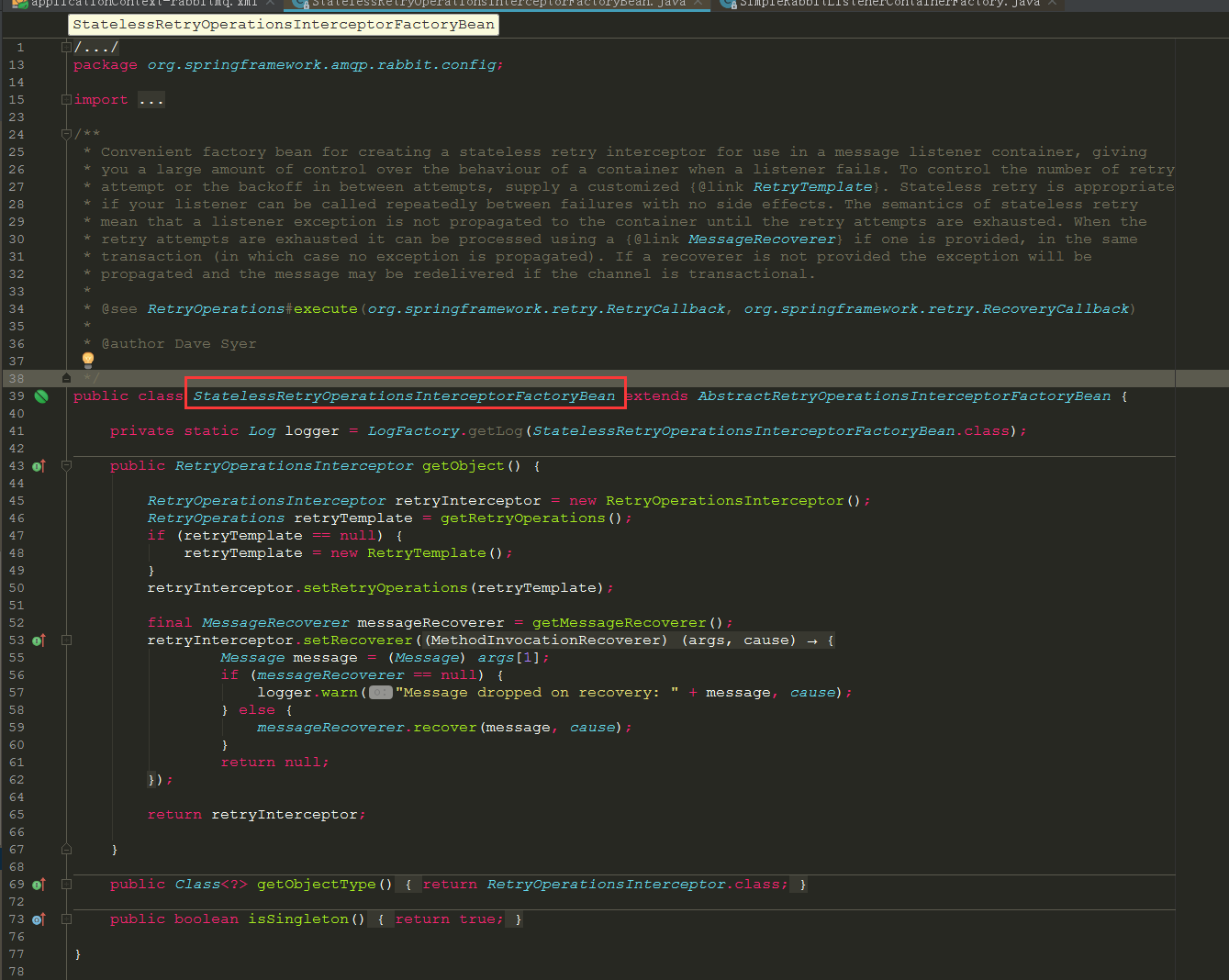
* 针对问题1，解决方案为：使用RabbitTemplate的事务参数：transactional



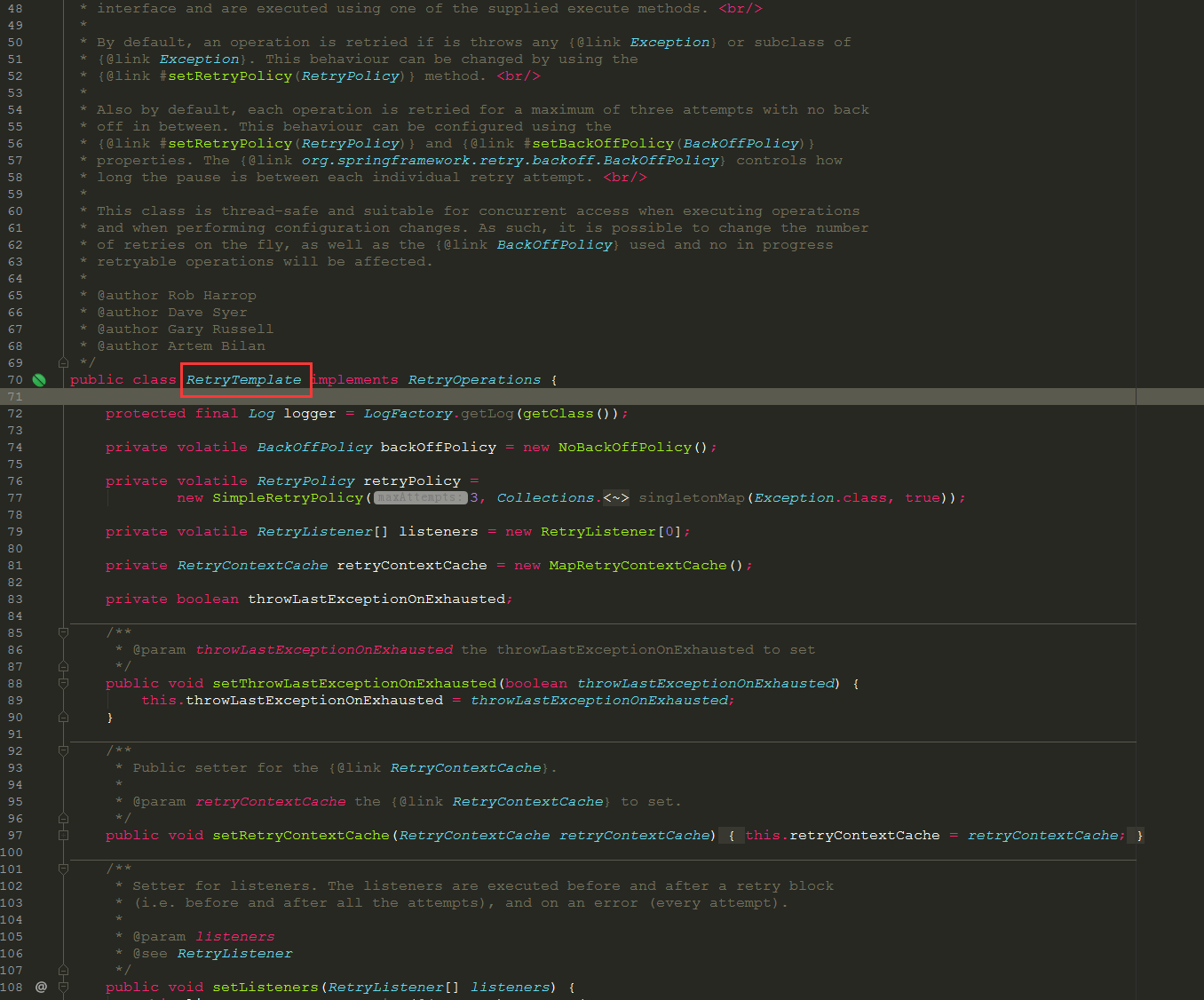
* 针对问题2，解决方案为：使用AbstractRabbitListenerContainerFactory的事务参数：channelTransacted



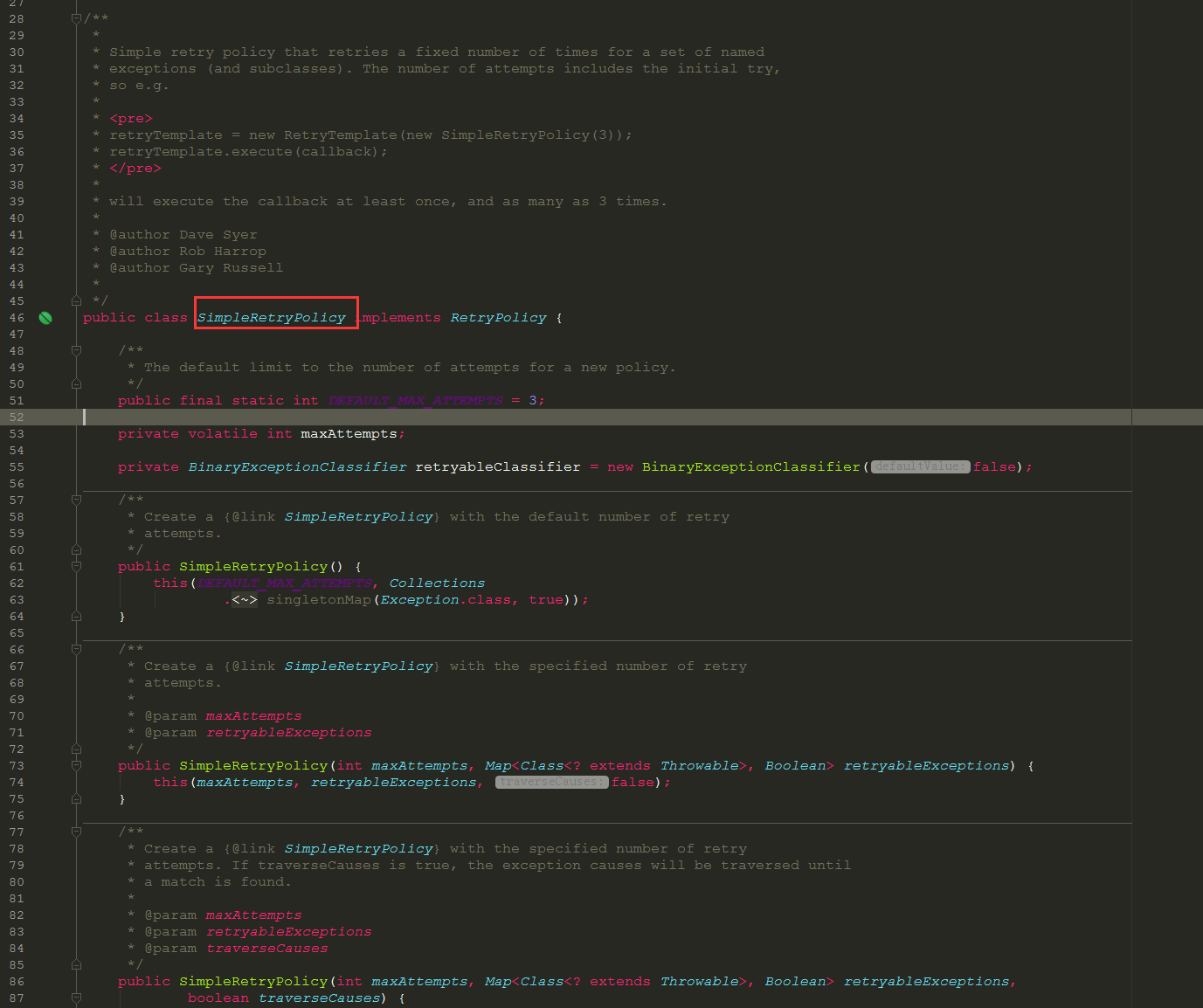
* 针对问题3，解决方案为：SimpleRabbitListenerContainerFactory中针对通知链adviceChain设置StatelessRetryOperationsInterceptorFactoryBean策略，解决重试问题。



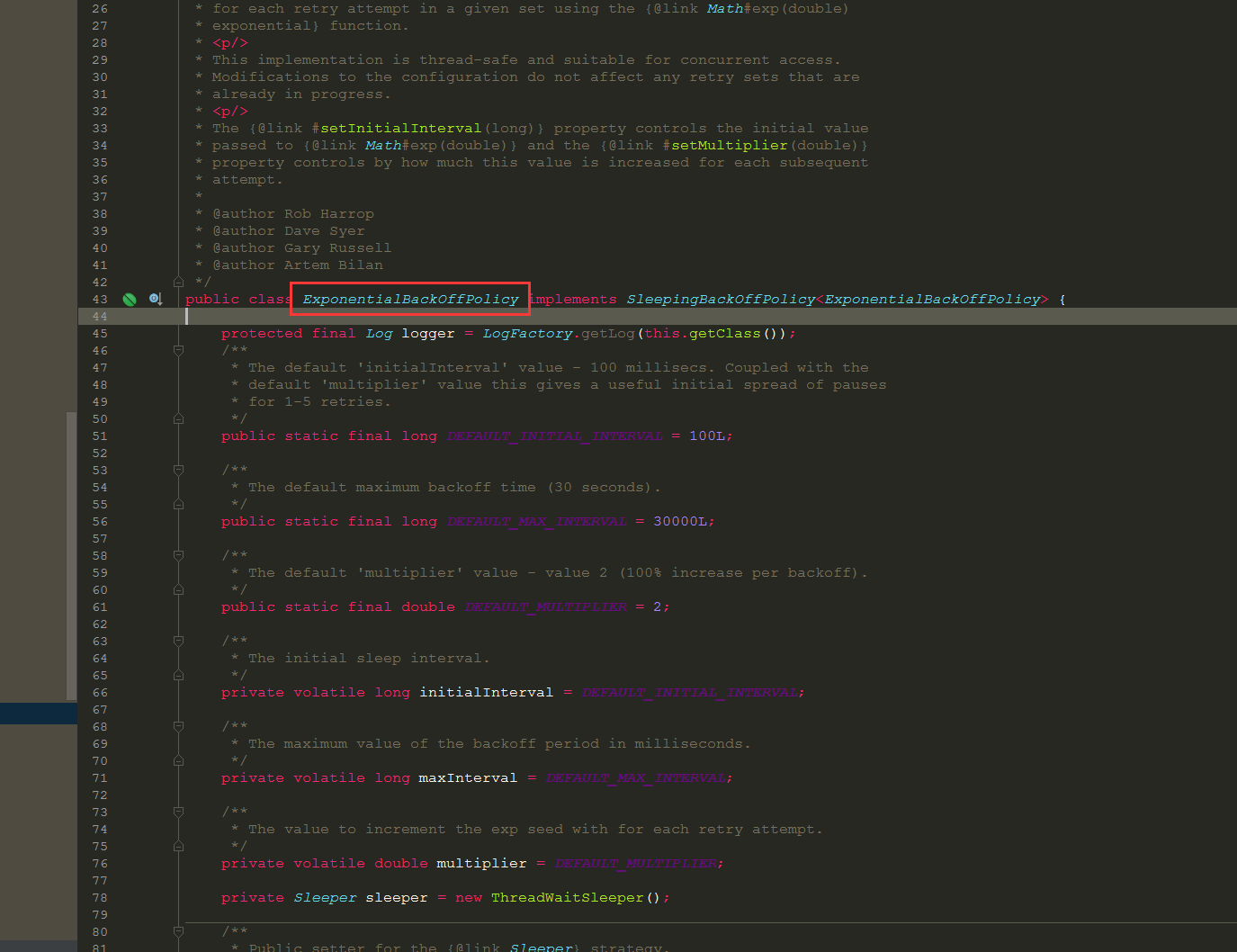
* 针对问题4，解决方案为：RetryPolicy + BackOffPolicy + Dead Letter Exchanges



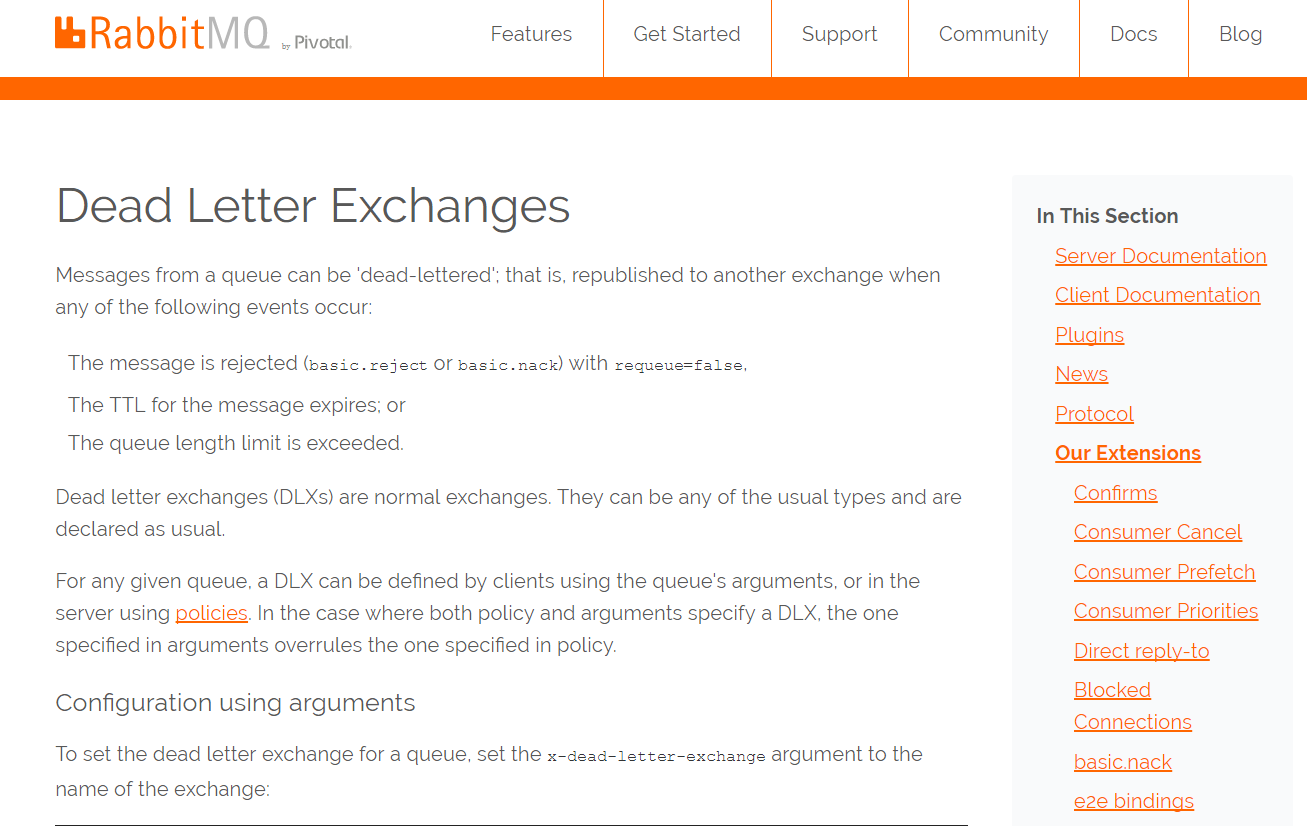
* **RetryTemplate**



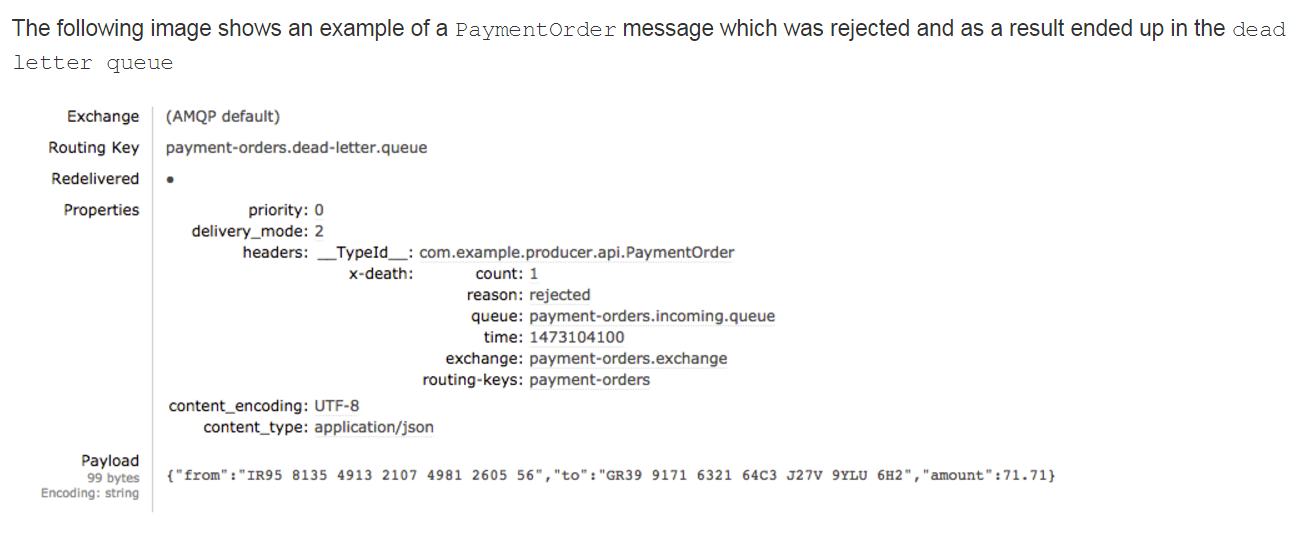
* **SimpleRetryPolicy**



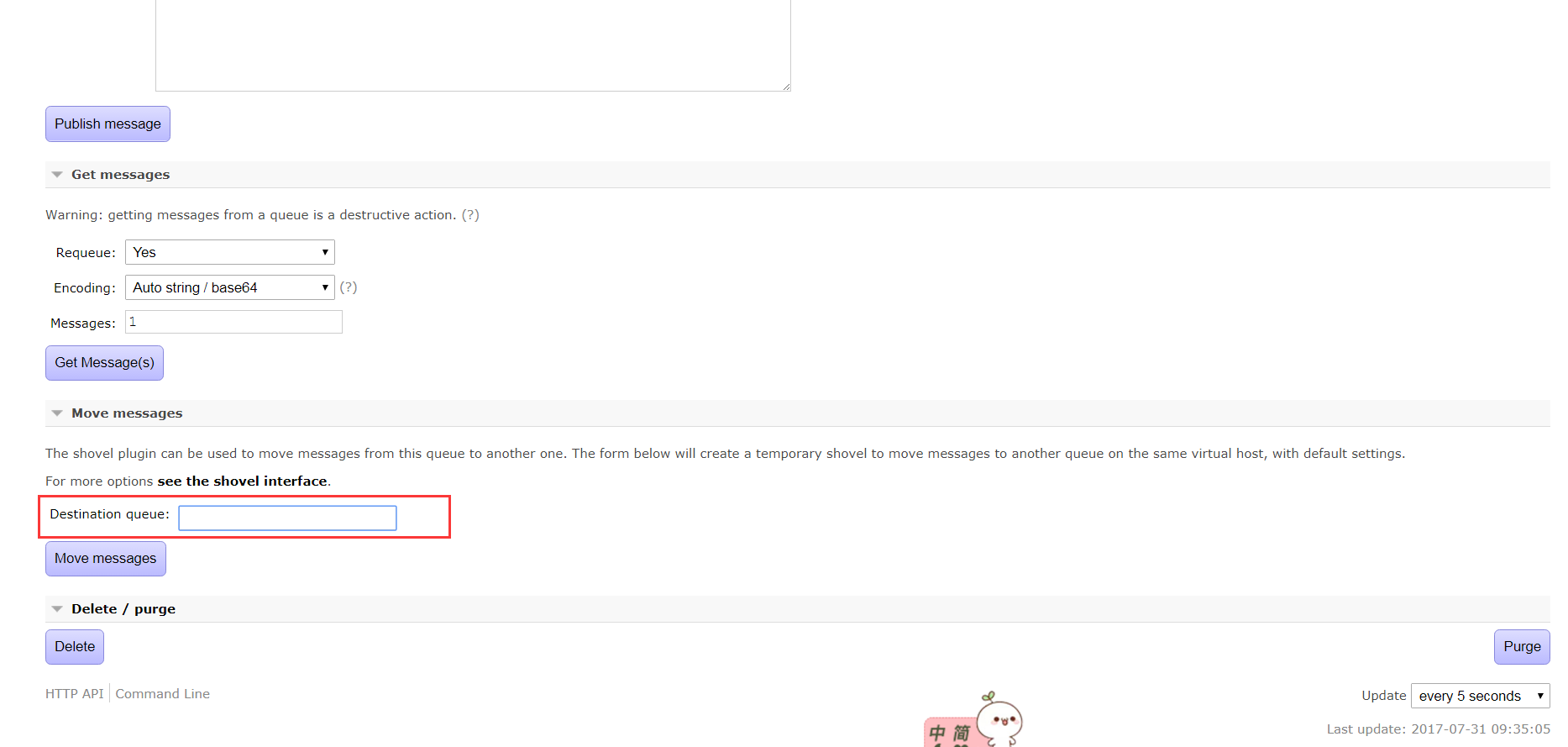
* **ExponentialBackOffPolicy**



* **Dead Letter Exchanges**



* **Dead Letter Queue**
* 针对问题5，解决方案为：使用RabbitMQ的管理界面插件实现消息转移，将Dead Letter Queue中的失败消息移动到正常的消费方监听队列中重新接收消息和处理即可。



## 消费方消息幂等性

重复消息是微服务实现中非常常见的问题，你永远不要指望调用方每次请求消息不一样，对于读操作，重复消息可能无害，可对于写操作很可能就是灾难。可以通过幂等(Idempotent)模式处理重复的消息，基本处理思路是：

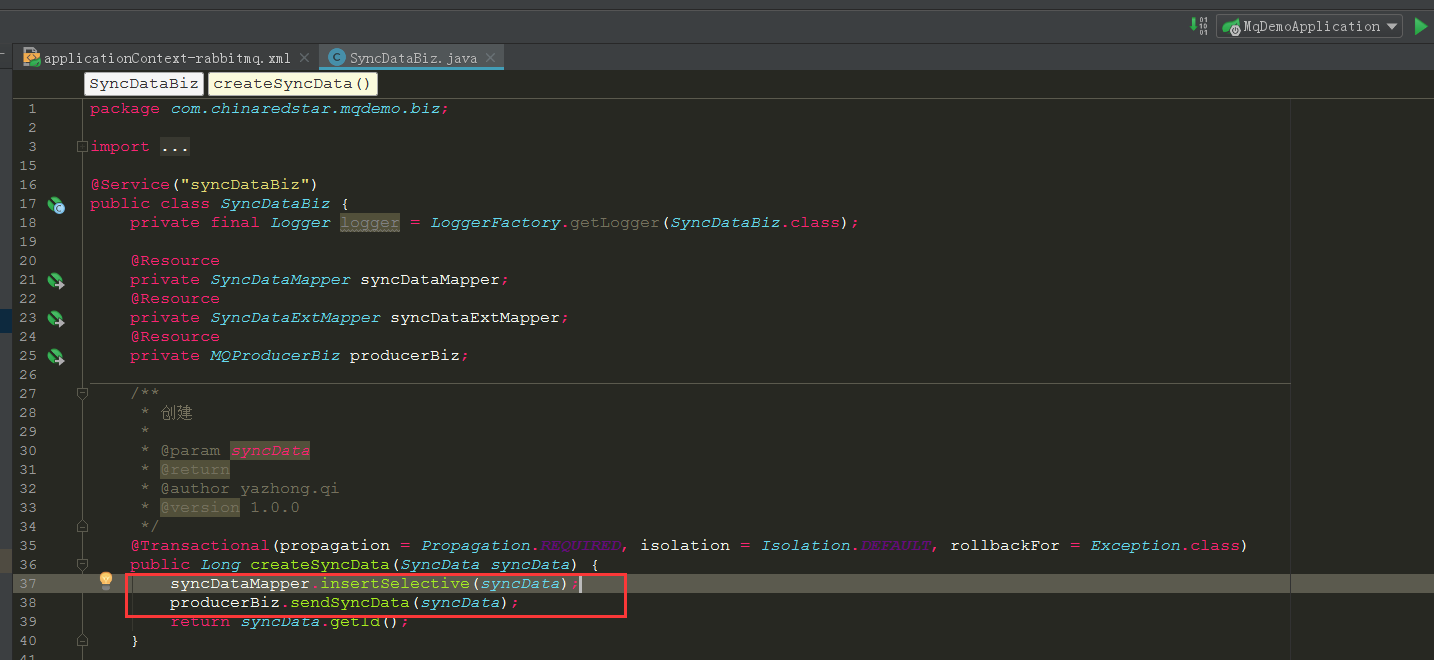
1、调用者给消息一个唯一请求ID标识。ID标识一个工作单元，这个工作单元只应执行一次，工作单元ID可以是Schema的一部分，也可以是一个定制的SOAP Header，服务的Contract 可以说明这个唯一请求ID标识是必须的；

2、接收者在执行一个工作单元必须先检验该工作单元是否已经执行过。检查是否执行的逻辑通常是根据唯一请求ID ，在服务端查询请求是否有记录，是否有对应的响应信息，如果有，直接把响应信息查询后返回；如果没有，那么就当做新请求去处理。

# 解决方案使用用例

针对mqDemo工程列举用例模型中的一些用例或场景。

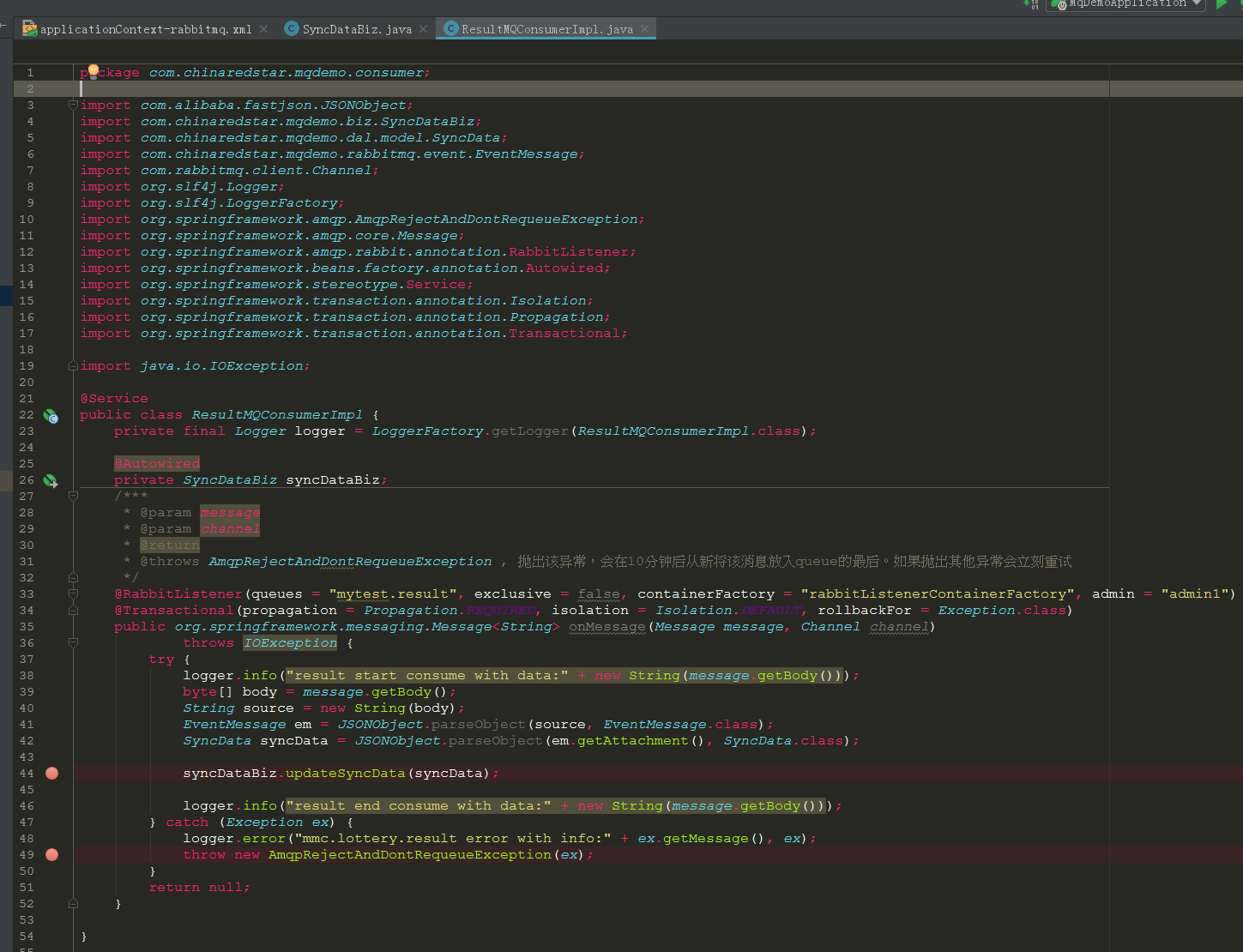
## 消息发送方用例



**数据库写操作和发送消息队列处于同一个事务之间**

**具体代码建demo程序。**

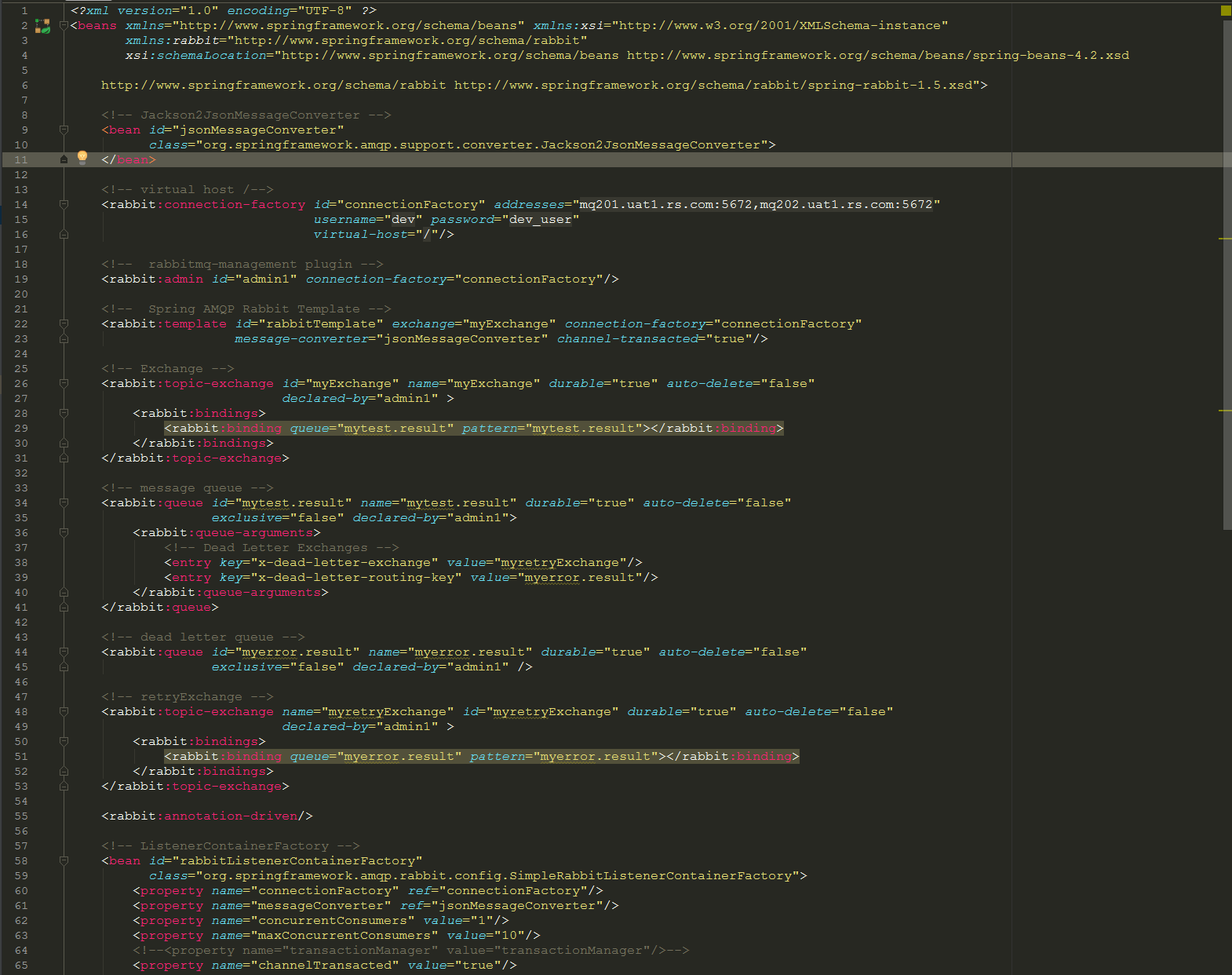
## 消息接收方用例

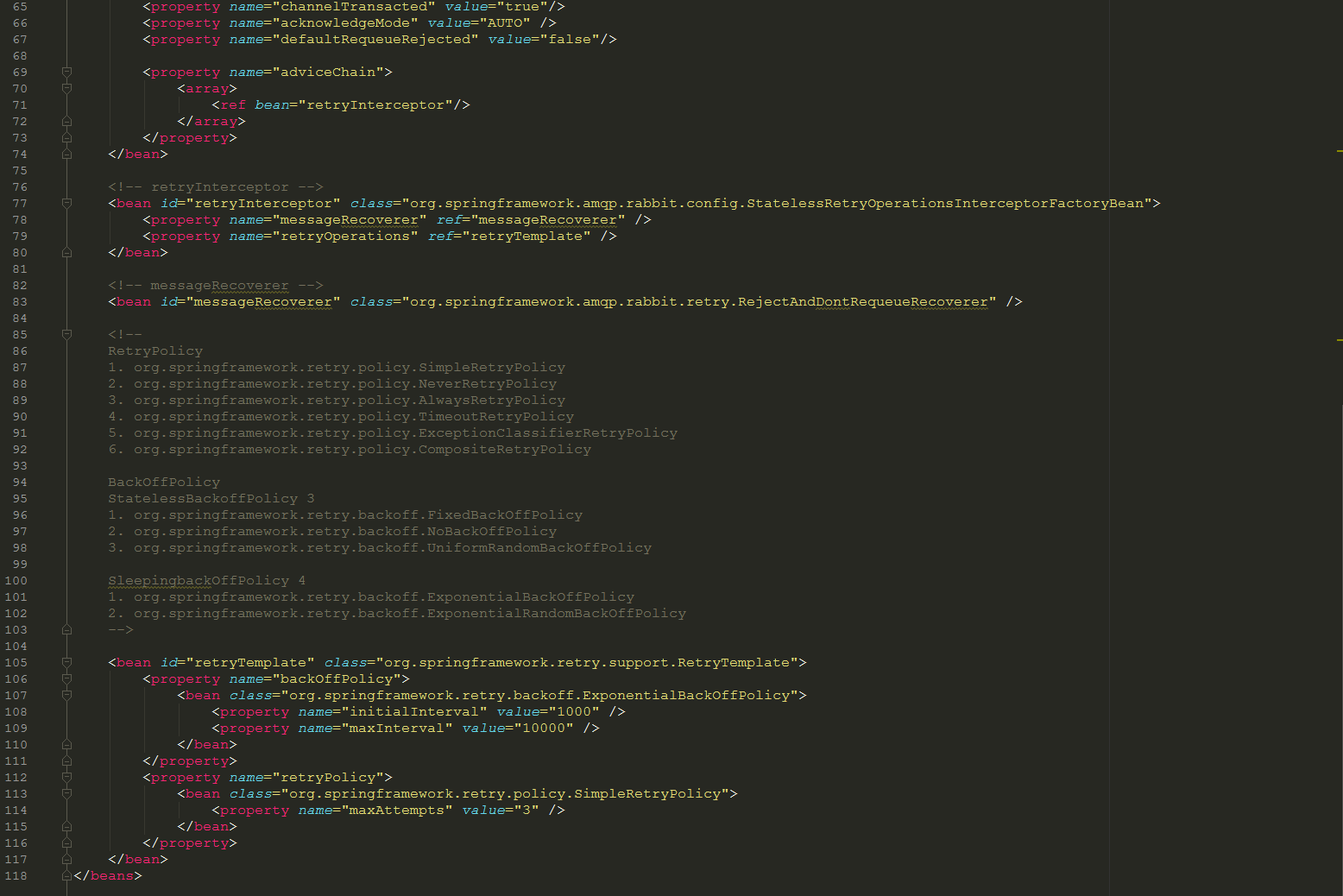


**接收消息，数据库本地写操作处于同一个事务之间**

**具体代码建demo程序。**

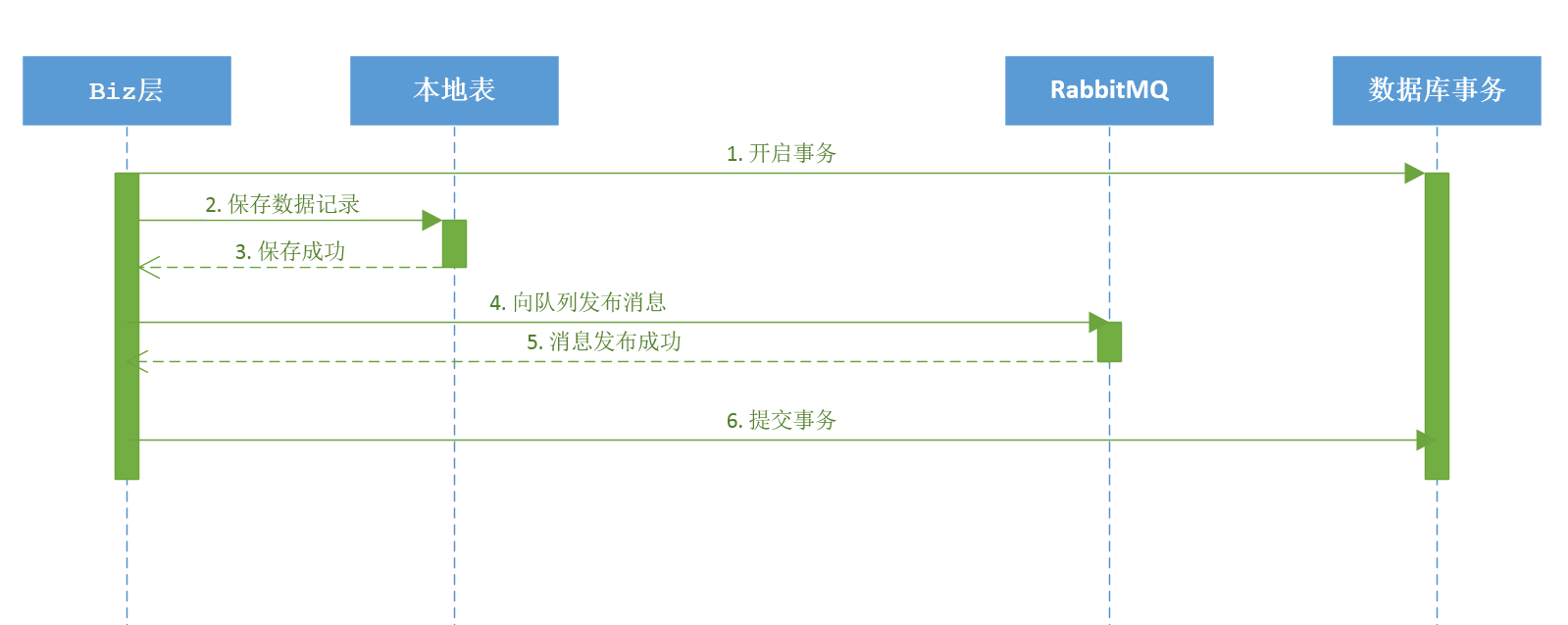
## 核心配置用例



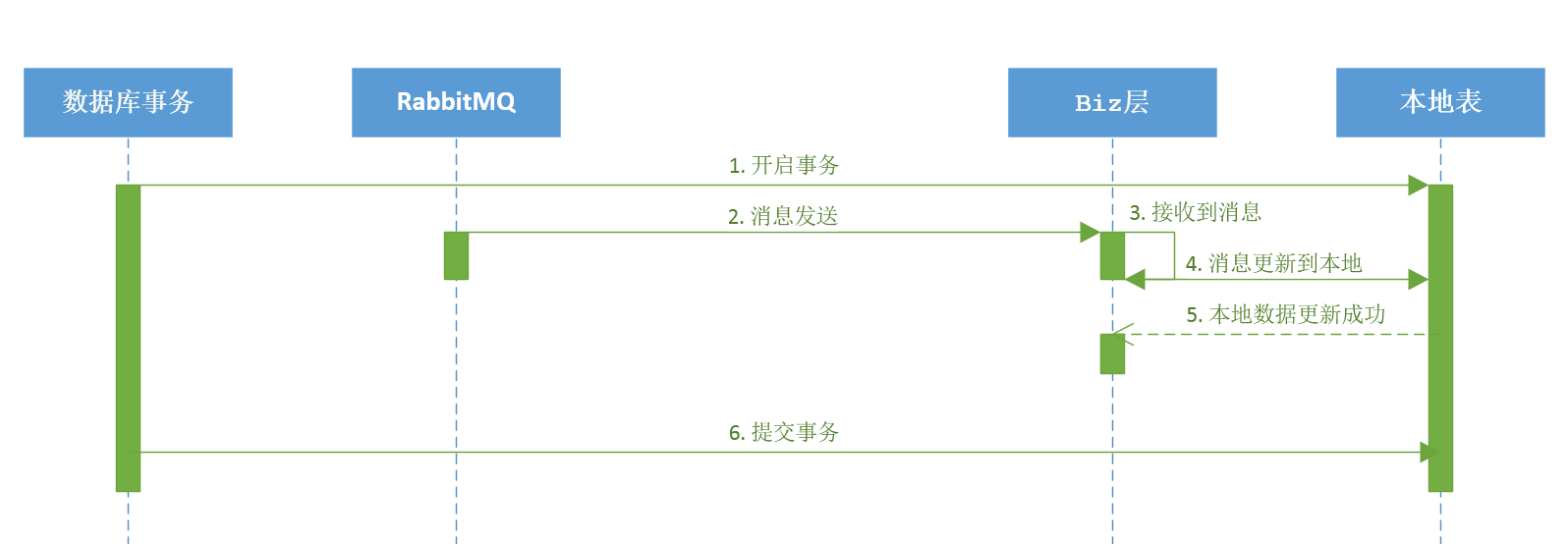


# 逻辑时序图说明

## 消息发送



## 消息接收



## 架构模式

架构采用的核心模式：消息驱动数据最终一致性

## 设计机制

在多个分布式事务处理方案：

* XA
* 2PC
* 3PC
* MQ message

最终选择最后一种。

# 性能

消息队列RabbitMQ拥有百万级别消息处理能力，但是由于使用时开启了事务特性，性能会有所下降，但是就目前龙翼系统数据交互量来讲，没有问题。

# 可扩展性

目前Demo中配置没有开启消费方异步多线程处理消息的功能，如果后期业务量增加可以尝试开始此设置，增加性能。

# 其它说明

暂无

# 附录A 指南

在此包括设计指南、编程指南、部署和安装指南，其中在编程指南上不仅包括传统的诸如《JAVA编程指南》，还包括指导实现人员实现特定设计机制要注意的问题

在此，推荐加入链接引用相关文档

# 附录B 规范

在此包括设计规范、编程规范等

在此，推荐加入链接引用相关文档