时间：2018年09月05日星期三

* 说明：本文部分内容均来自慕课网。
* @慕课网：[https://www.imooc.com](https://www.imooc.com/)
* 教学源码：无
* 学习源码：https://github.com/zccodere/study-imooc

# **第一章：RabbitMQ起步**

## **1-1 课程导航**

* RabbitMQ简介及AMQP协议
* RabbitMQ安装与使用
* RabbitMQ核心概念
* 与SpringBoot整合
* 保障100%的消息可靠性投递方案落地实现

## **1-2 RabbitMQ简介**

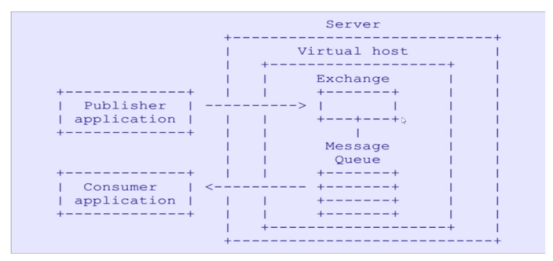
### 1.2.1 初识RabbitMQ

* RabbitMQ是一个开源的消息代理和队列服务器
* 用来通过普通协议在完全不同的应用之间共享数据
* RabbitMQ是使用Erlang语言来编写的
* 并且RabbitMQ是基于AMQP协议的

### 1.2.2 RabbitMQ简介

* 目前很多互联网大厂都在使用RabbitMQ
* RabbitMQ底层采用Erlang语言进行编写
* 开源、性能优秀，稳定性保障
* 与SpringAMQP完美的整合、API丰富
* 集群模式丰富，表达式配置，HA模式，镜像队列模型
* 保证数据不丢失的前提做到高可靠性、可用性
* AMQP全称：Advanced Message Queuing Protocol
* AMQP翻译：高级消息队列协议

### 1.2.3 AMQP协议模型



**Server:** 又称Broker,接受客户端的连接,实现AMQP实体服务

**Connection:** 连接,应用程序与Broker的网络连接

**Channel:** 网络信道,几乎所有的操作都在channel中进行,Channel是进行消息读写的通道.客户端可建立多个Channel,每个Channel代表一个会话任务.

**Message:** 消息,服务器和应用程序之前的传送的数据,由Properties和Body组成.Properites可以对消息进行修饰(设置),比如消息的优先级,延迟等高级特性;

Body就是消息体内容.

**Virtual host:** 虚拟地址,用于进行逻辑隔离,最上层的消息路由.一个Virtual host里面可以有若干个Exchange和Queue,同一个Virtual host里面不能有相同的Exchange或Queue

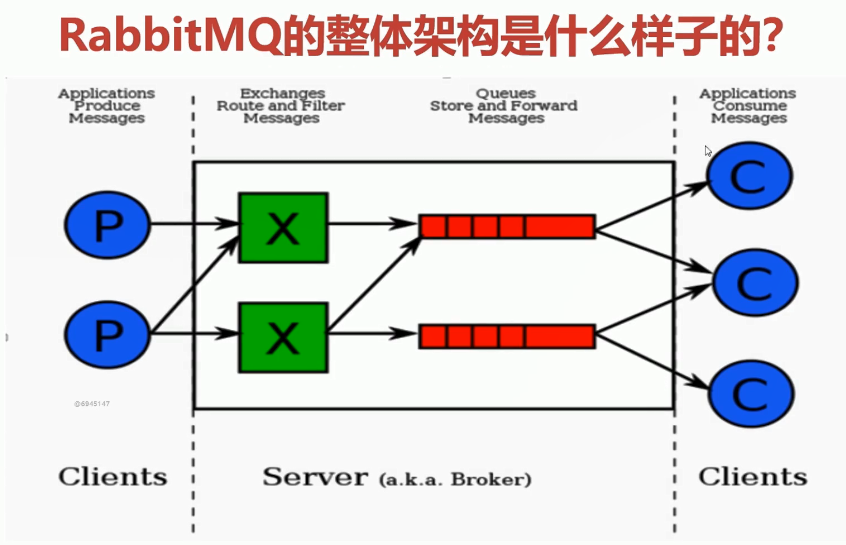
**Exchange:** 交换机,接收消息,根据路由键转发消息到绑定的队列

**Binding:** Exchange和Queue只见的虚拟连接,binding中可以包含routing key

**Routing key:** 一个路由规则,虚拟机可以用它来确定如何路由一个特定消息

**Queue:** 也称为Message Queue,消息队列,保存消息并将他们转发给消费者

### 1.2.4 RabbitMQ的整理架构



1. 生产者将消息发送到exchange交换机上
2. exchange交换机经过路由和过滤将消息发送到指定的队列中
3. 消费者从指定的队列中进行消费消息

## **1-3 RabbitMQ安装**

### 环境安装可参考其他安装文档

### 0.安装准备

官网地址：http://www.rabbitmq.**com**/

安装Linux必要依赖包<Linux7>

下载RabbitMQ安装包

进行安装，修改相关配置文件

**vim** /etc/hostname

**vim** /etc/hosts

### 1.安装Erlang

wget https://packages.erlang-solutions.**com**/erlang-solutions\_1.0\_all.**deb**

sudo dpkg -i erlang-solutions\_1.0\_all.**deb**

sudo apt-get install erlang erlang-nox

### 安装RabbitMQ

**echo** 'deb http://www.rabbitmq.com/debian/ testing main' | sudo tee /etc/apt/sources.**list**.d/rabbitmq.**list**

wget -O- https://www.rabbitmq.**com**/rabbitmq-release-signing-key.asc | sudo apt-key add -

sudo apt-get install rabbitmq-server

### 3.安装RabbitMQ web管理插件

sudo rabbitmq-plugins enable rabbitmq\_management

sudo systemctl restart rabbitmq-server

访问：http://localhost:15672

默认用户名密码：guest/guest

### 4.修改RabbitMQ配置

**vim** /usr/lib/rabbitmq/lib/rabbitmq\_server-3.5.7/ebin/rabbit.app

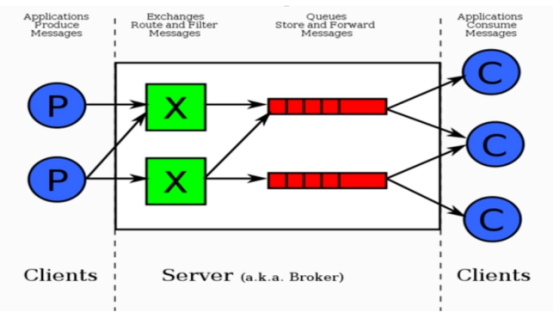
比如修改密码、配置等等；例如：loopback\_users中的<<"guest">>，只保留guest

服务启动：rabbitmq-server start &

服务停止：rabbitmqctl app\_stop

## **1-4 RabbitMQ概念**

### 1.4.1 RabbitMQ的整体架构



### 1.4.2 RabbitMQ核心概念

* Server：又称Broker，接受客户端的连接，实现AMQP实体服务
* Connection：连接，应用程序与Broker的网络连接
* Channel：网络信道

1. 几乎所有的操作都在Channel中进行
2. Channel是进行消息读写的通道
3. 客户端可建立多个Channel
4. 每个Channel代表一个会话任务

* Message：消息

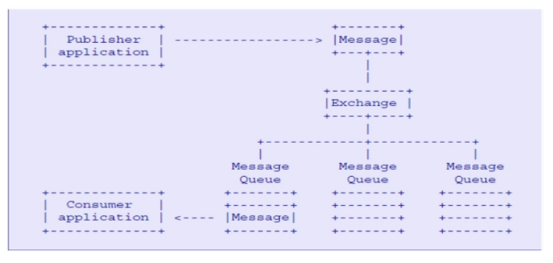
服务器和应用程序之间传送的数据，由Properties和Body组成  
Properties可以对消息进行修饰，比如消息的优先级、延迟等高级特性  
Body则就是消息体内容

* Virtual host：虚拟机

用于进行逻辑隔离，最上层的消息路由一个Virtual host里面可以有若干个Exchange和Queue同一个Virtual host里面不能有相同名称的Exchange或Queue

* Exchange：交换机，接收消息，根据路由键转发消息到绑定的队列
* Binding：Exchange和Queue之间的虚拟连接，binding中可以包含routing key
* Routing key：一个路由规则，虚拟机可用它来确定如何路由一个特定消息
* Queue：也称为Message Queue，消息队列，保存消息并将它们转发给消费者

### 1.4.3 RabbitMQ消息的流转过程



## 1-5 RabbitMQ命令行与管控台

rabbitmqctl命令是最丰富的

rabbitmqctl stop\_app:关闭应用

rabbitmqctl start\_app:启动应用

rabbitmqctl status:查看节点状态

**操作用户:**

rabbitmqctl add\_user username password:添加用户

rabbitmqctl list\_users:列出所有用户

rabbitmqctl delete\_user username:删除用户

rabbitmqctl clear\_permissions -p vhostpath username:清除用户权限

rabbitmqctl list\_user\_permissions username:列出用户权限

rabbitmqctl change\_password username newpassword:修改用户密码

rabbitmqctl set\_permissions -p vhostpath username “.\*” “.\*” “.\*”:设置用户权限

**操作虚拟主机:**

rabbitmqctl add\_vhost vhostpath:创建虚拟主机

rabbitmqctl list\_vhosts:列出所有虚拟主机

rabbitmqctl list\_permissions -p vhostpath:列出所有虚拟主机上所有权限

rabbitmqctl delete\_vhost vhostpath:删除虚拟主机

**操作队列:**

rabbitmqctl list\_queues:查看所有队列信息

rabbitmqctl -p vhostpath purge\_queue blue:清楚队列里的消息

**操作集群:**

rabbitmqctl reset: 移除所有数据,要在rabbitmqctl stop\_app之后

rabbitmqctl join\_cluster <clusternode> [--ram]:组成集群

rabbitmqctl cluster\_status:查看集群状态

rabbitmqctl change\_cluster\_node\_type disc | ram 修改集群节点的存储形式

rabbitmqctl forget\_cluster\_node [--offline] 忘记节点(摘除节点)

rabbitmqctl rename\_cluster\_node oldnode1 newnode1 修改节点名称

rabbitmq的版本数据迁移的时候使用导入导出工具进行导出配置文件等信息

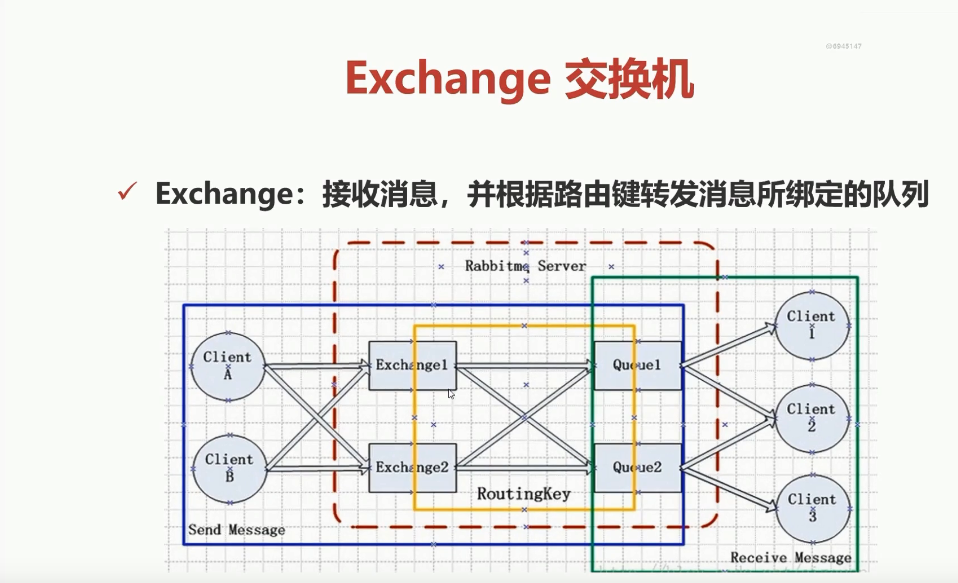
## 1-6 急速入门-消息生产与消费

1) 创建ConnectionFactory:获取连接工厂

2) Connection:创建一个连接

3) 通过连接创建Channel:数据通信信道,可发送和接收消息

## 1-7 交换机详解



### 交换机属性

**Name:** 交换机名称

**Type:** 交换机类型 direct,topic,fanout,headers

**Durability:** 是否需要持久化,true为持久化

**Auto Delete:** 当最后一个绑定到Exchange上队列被删除之后,自动删除该Exchange

**Internal:** 当前Exchange是否用于RabbitMQ内部使用,默认false(99%的情况)

**Arguments:** 扩展参数,用于扩展AMQP协议自制定化使用

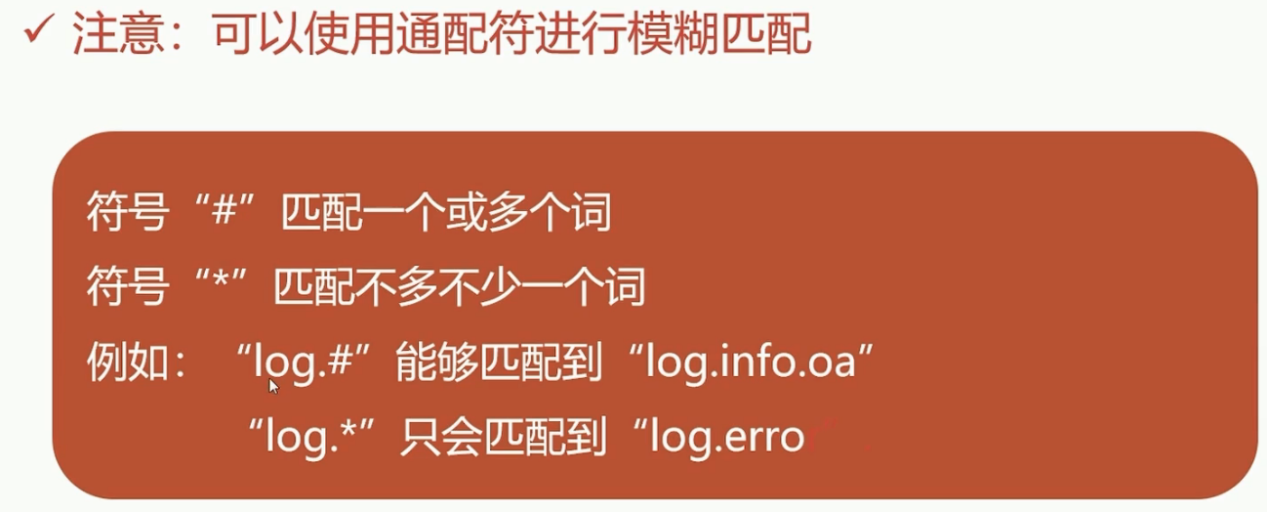
### Direct Exchange

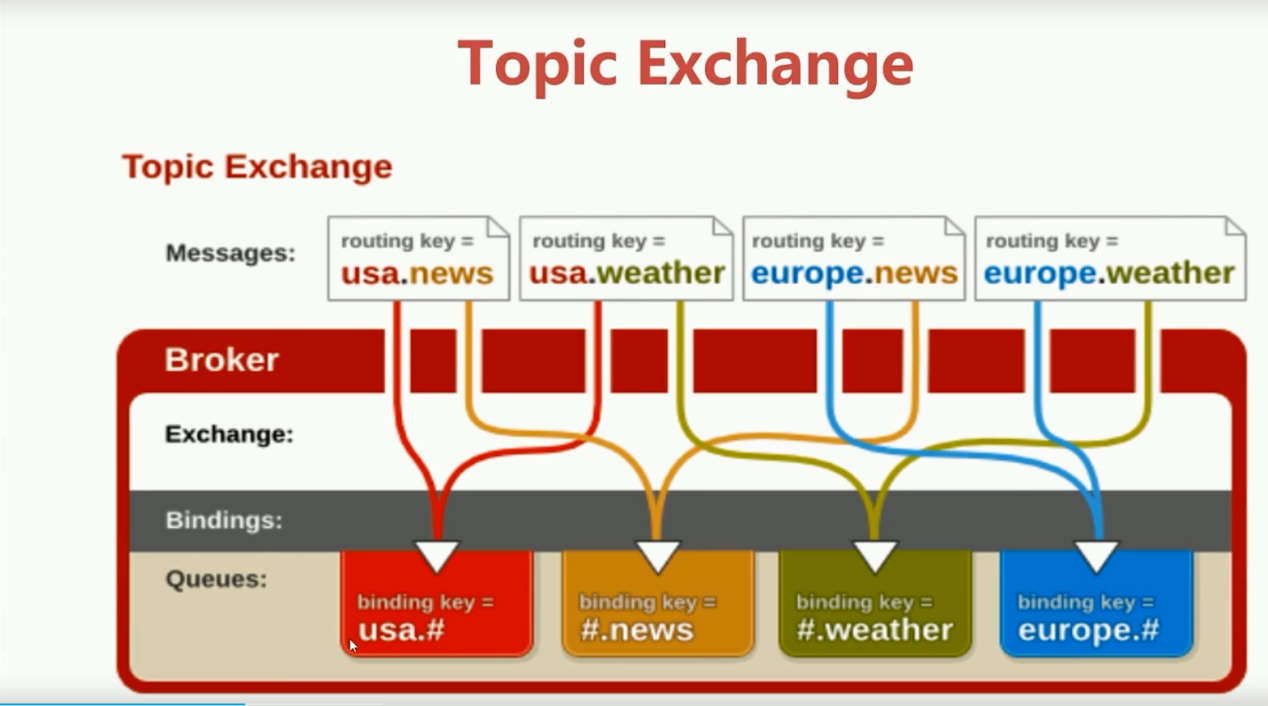
所有发送到Direct Exchange的消息被转发到Routekey中指定的Queue

注意:Direct模式可以使用RabbitMQ自带的Exchange:default Exchange,所以不需要将Exchange进行任何的绑定操作,消息传递时,RouteKey必须完全匹配才会被队列接收,否则该消息会被抛弃.

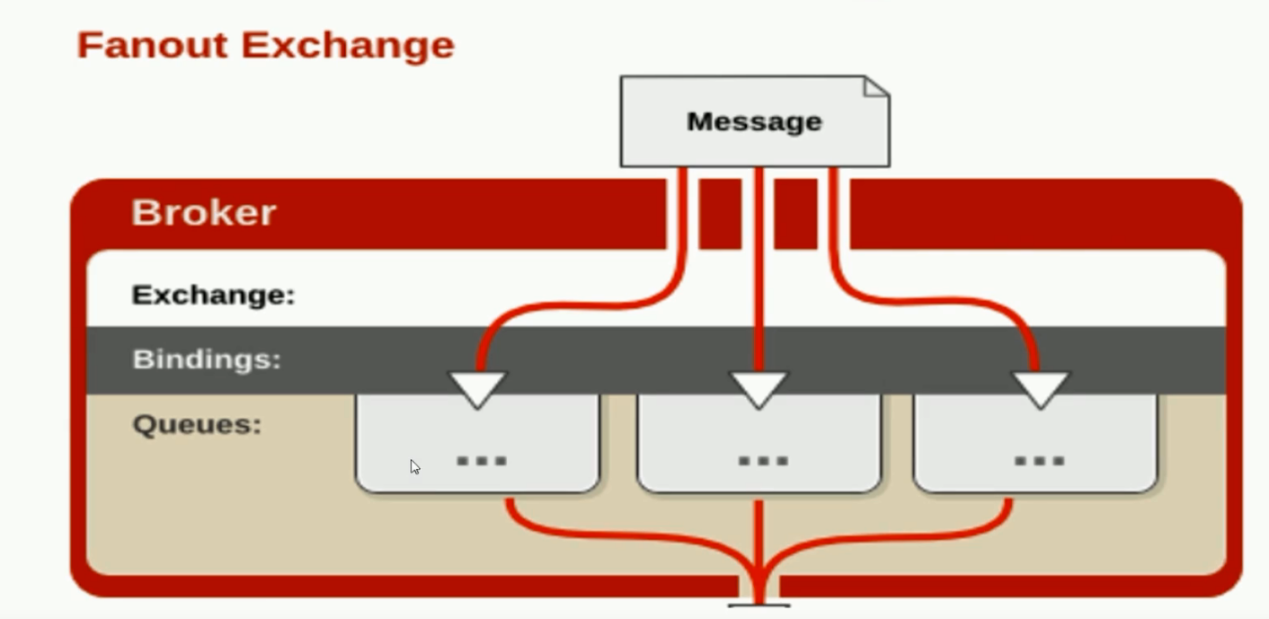
### Topic Exchange

* 所有发送到Topic Exchange 的消息被转发到所有关心Routekey中指定Topic的Queue上
* Exchange将RouteKey和某Topic进行模糊匹配，此时队列需要绑定一个Topic





### Fanout Exchange(常用)



* 不处理路由键，只需要简单的将队列绑定到指定的交换机上
* 发送到交换机的消息都会被转发到与该交换机绑定的所有队列上
* Fanout交换机转发消息是最快的

## 1-8 绑定、队列、消息、虚拟主机小结

### Binding-绑定

* Exchange和Exchange、Queue之间的连接关系
* Binding中可以包含RoutingKey或者参数

### Queue-消息队列

* 消息队列，实际存储消息数据
* Durability：是否持久化，Durable:是，Transient：否
* Auto delete:如选yes，代表当最后一个监听被移除之后，该Queue会自动被删除

### Message-消息

* 服务器和应用程序之间传送的数据
* 本质上就是一段数据，由Properties和Payload（Body）组成
* 常用属性：delivery mode、headers（自定义属性）

### Message-其他属性

* content\_type、content\_encoding、priority
* correlation\_id、reply\_to、expiration、message\_id
* timestamp、type、user\_id、app\_id、cluster\_id

### Virtual host-虚拟主机

* 虚拟地址，用于进行逻辑隔离，最上层的消息路由
* 一个Virtual Host里面可以有若干个Exchange和Queue
* 同一个Virtual Host里面不能有相同名称的Exchange或Queue

# **第二章：RabbitMQ使用**

## **2-1 发送消息**

**SpringBoot与RabbitMQ集成**

* 引入相关依赖
* 对application.properties进行配置

创建名为rabbitmq-producer的maven工程pom如下

**<?**xml version="1.0" encoding="UTF-8"**?>**<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>47-rabbitmq</artifactId>

<groupId>com.myimooc</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>rabbitmq-producer</artifactId>

<properties>

<spring.boot.version>2.0.4.RELEASE</spring.boot.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-parent</artifactId>

<version>${spring.boot.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

*<!--RabbitMQ依赖-->*

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

*<!--工具类依赖-->*

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.5</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.36</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

1.编写Order类

package com.myimooc.rabbitmq.entity;

**import** java.io.Serializable;

*/\*\**

*\* <br>*

*\* 标题: 订单实体<br>*

*\* 描述: 订单实体<br>*

*\* 时间: 2018/09/06<br>*

*\**

*\* @author zc*

*\*/***public** **class** Order implements Serializable{

**private** **static** final **long** serialVersionUID = 6771608755338249746L;

**private** String id;

**private** String name;

*/\*\**

*\* 存储消息发送的唯一标识*

*\*/*

**private** String messageId;

**public** **Order**() {

}

**public** **Order**(String id, String name, String messageId) {

**this**.id = id;

**this**.name = name;

**this**.messageId = messageId;

}

@Override

**public** String **toString**() {

**return** "Order{" +

"id='" + id + ''' +

", name='" + name + ''' +

", messageId='" + messageId + ''' +

'}';

}

**public** String **getId**() {

**return** id;

}

**public** **void** **setId**(String id) {

**this**.id = id;

}

**public** String **getName**() {

**return** name;

}

**public** **void** **setName**(String name) {

**this**.name = name;

}

**public** String **getMessageId**() {

**return** messageId;

}

**public** **void** **setMessageId**(String messageId) {

**this**.messageId = messageId;

}

}

2.编写OrderSender类

**package** com.myimooc.rabbitmq.producer.producer;

**import** com.myimooc.rabbitmq.entity.Order;**import** org.springframework.amqp.rabbit.core.RabbitTemplate;**import** org.springframework.amqp.rabbit.support.CorrelationData;**import** org.springframework.beans.factory.annotation.Autowired;**import** org.springframework.stereotype.Component;

*/\*\**

*\* <br>*

*\* 标题: 订单消息发送者<br>*

*\* 描述: 订单消息发送者<br>*

*\* 时间: 2018/09/06<br>*

*\**

*\* @author zc*

*\*/***@Component**

**public** **class** **OrderSender** {

**private** RabbitTemplate rabbitTemplate;

**@Autowired**

**public** **OrderSender**(

RabbitTemplate rabbitTemplate) {

**this**.rabbitTemplate = rabbitTemplate;

}

*/\*\**

*\* 发送订单*

*\**

*\* @param order 订单*

*\* @throws Exception 异常*

*\*/*

**public** **void** **send**(Order order) **throws** Exception {

CorrelationData correlationData = **new** CorrelationData();

correlationData.setId(order.getMessageId());

*// exchange：交换机*

*// routingKey：路由键*

*// message：消息体内容*

*// correlationData：消息唯一ID*

**this**.rabbitTemplate.convertAndSend("order-exchange", "order.a", order, correlationData);

}

}

3.编写application.properties类

*# RabbitMQ配置*

spring.rabbitmq.addresses=192.168.0.105:5672

spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

spring.rabbitmq.virtual-host=/

spring.rabbitmq.connection-timeout=15000

*# Server配置*

server.servlet.context-path=/

server.port=8080

spring.http.encoding.charset=UTF-8

spring.jackson.date-format=yyyy-MM-dd HH:mm:ss

spring.jackson.time-zone=GMT+8

spring.jackson.default-property-inclusion=NON\_NULL

4.编写Application类

**package** com.myimooc.rabbitmq.producer;

**import** org.springframework.boot.SpringApplication;**import** org.springframework.boot.autoconfigure.SpringBootApplication;

*/\*\**

*\* <br>*

*\* 标题: 启动类<br>*

*\* 描述: 启动类<br>*

*\* 时间: 2018/09/06<br>*

*\**

*\* @author zc*

*\*/***@SpringBootApplicationpublic** **class** **Application** {

**public** **static** **void** main(String[] args) {

SpringApplication.run(Application.**class**, args);

}

}

5.编写OrderSenderTest类

**package** com.myimooc.rabbitmq.producer.producer;

**import** com.myimooc.rabbitmq.entity.Order;**import** org.junit.Test;**import** org.junit.runner.RunWith;**import** org.springframework.beans.factory.annotation.Autowired;**import** org.springframework.boot.test.context.SpringBootTest;**import** org.springframework.test.context.junit4.SpringRunner;

**import** java.util.UUID;

*/\*\**

*\* <br>*

*\* 标题: 订单消息发送者测试<br>*

*\* 描述: 订单消息发送者测试<br>*

*\* 时间: 2018/09/06<br>*

*\**

*\* @author zc*

*\*/***@RunWith**(SpringRunner.**class**)**@SpringBootTestpublic** **class** **OrderSenderTest** {

**@Autowired**

**private** OrderSender orderSender;

**@Test**

**public** **void** testSend1() **throws** Exception {

Order order = **new** Order();

order.setId("201809062009010001");

order.setName("测试订单1");

order.setMessageId(System.currentTimeMillis() + "$" + UUID.randomUUID().toString().replaceAll("-",""));

**this**.orderSender.send(order);

}

}

## **2-2 处理消息**

创建名为rabbitmq-consumer的maven工程pom如下

**<?**xml version="1.0" encoding="UTF-8"**?>**<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<parent>

<artifactId>47-rabbitmq</artifactId>

<groupId>com.myimooc</groupId>

<version>1.0-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>rabbitmq-consumer</artifactId>

<properties>

<spring.boot.version>2.0.4.RELEASE</spring.boot.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-parent</artifactId>

<version>${spring.boot.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

*<!--RabbitMQ依赖-->*

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

*<!--工具类依赖-->*

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.5</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.36</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</dependency>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

1.编写Order类

package com.myimooc.rabbitmq.entity;

**import** java.io.Serializable;

*/\*\**

*\* <br>*

*\* 标题: 订单实体<br>*

*\* 描述: 订单实体<br>*

*\* 时间: 2018/09/06<br>*

*\**

*\* @author zc*

*\*/***public** **class** Order implements Serializable{

**private** **static** final **long** serialVersionUID = 6771608755338249746L;

**private** String id;

**private** String name;

*/\*\**

*\* 存储消息发送的唯一标识*

*\*/*

**private** String messageId;

**public** **Order**() {

}

**public** **Order**(String id, String name, String messageId) {

**this**.id = id;

**this**.name = name;

**this**.messageId = messageId;

}

@Override

**public** String **toString**() {

**return** "Order{" +

"id='" + id + ''' +

", name='" + name + ''' +

", messageId='" + messageId + ''' +

'}';

}

**public** String **getId**() {

**return** id;

}

**public** **void** **setId**(String id) {

**this**.id = id;

}

**public** String **getName**() {

**return** name;

}

**public** **void** **setName**(String name) {

**this**.name = name;

}

**public** String **getMessageId**() {

**return** messageId;

}

**public** **void** **setMessageId**(String messageId) {

**this**.messageId = messageId;

}

}

2.编写OrderReceiver类

**package** com.myimooc.rabbitmq.consumer.consumer;

**import** com.rabbitmq.client.Channel;**import** com.myimooc.rabbitmq.entity.Order;**import** org.springframework.amqp.rabbit.**annotation**.\*;**import** org.springframework.amqp.support.AmqpHeaders;**import** org.springframework.messaging.handler.**annotation**.Headers;**import** org.springframework.messaging.handler.**annotation**.Payload;**import** org.springframework.stereotype.Component;

**import** java.util.Map;

*/\*\**

*\* <br>*

*\* 标题: 订单接收者<br>*

*\* 描述: 订单接收者<br>*

*\* 时间: 2018/09/06<br>*

*\**

*\* @author zc*

*\*/***@Componentpublic** **class** **OrderReceiver** {

*/\*\**

*\* 接收消息*

*\**

*\* @param order 消息体内容*

*\* @param headers 消息头内容*

*\* @param channel 网络信道*

*\* @throws Exception 异常*

*\*/*

**@RabbitListener(bindings = @QueueBinding(**

**value = @Queue(value = "order-queue",durable = "true")**,

exchange = **@Exchange(name = "order-exchange",type = "topic")**,

key = "order.\*"

))

**@RabbitHandler**

**public** void onOrderMessage(**@Payload** Order order, **@Headers** Map<String, Object> headers, Channel channel) throws Exception {

*// 消费者操作*

System.**out**.println("收到消息：");

System.**out**.println("订单信息：" + order.toString());

*// 手动签收消息*

Long deliveryTag = (Long) headers.**get**(AmqpHeaders.DELIVERY\_TAG);

channel.basicAck(deliveryTag, false);

}

}

3.编写application.properties类

*# RabbitMQ连接配置*

spring.rabbitmq.addresses=192.168.0.105:5672

spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

spring.rabbitmq.virtual-host=/

spring.rabbitmq.connection-timeout=15000*# RabbitMQ消费配置# 基本并发：5*

spring.rabbitmq.listener.simple.concurrency=5*# 最大并发：10*

spring.rabbitmq.listener.simple.max-concurrency=10*# 签收模式：手动签收*

spring.rabbitmq.listener.simple.acknowledge-mode=manual*# 限流策略：同一时间只有1条消息发送过来消费*

spring.rabbitmq.listener.simple.prefetch=1

*# Server配置*

server.servlet.context-path=/

server.port=8082

spring.http.encoding.charset=UTF-8

spring.jackson.date-format=yyyy-MM-dd HH:mm:ss

spring.jackson.time-zone=GMT+8

spring.jackson.default-property-inclusion=NON\_NULL

4.编写Application类

**package** com.myimooc.rabbitmq.consumer;

**import** org.springframework.boot.SpringApplication;**import** org.springframework.boot.autoconfigure.SpringBootApplication;

*/\*\**

*\* <br>*

*\* 标题: 启动类<br>*

*\* 描述: 启动类<br>*

*\* 时间: 2018/09/06<br>*

*\**

*\* @author zc*

*\*/***@SpringBootApplicationpublic** **class** **Application** {

**public** **static** **void** main(String[] args) {

SpringApplication.run(Application.**class**, args);

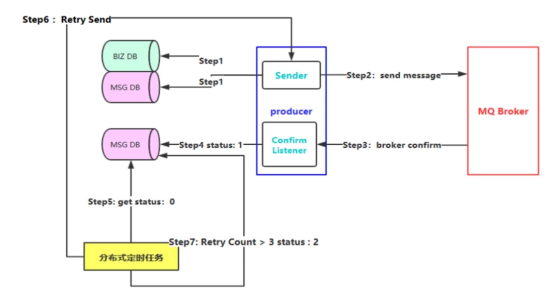
}

}

# **第三章：可靠性投递**

## **3-1 设计方案**

保障100%消息投递成功设计方案（一）



## **3-2 代码详解**

因篇幅限制，源码请到github地址查看，这里仅展示核心关键类

1.编写OrderSender类

**package** com.myimooc.rabbitmq.ha.producer;

**import** com.myimooc.rabbitmq.entity.Order;**import** com.myimooc.rabbitmq.ha.constant.Constants;**import** com.myimooc.rabbitmq.ha.dao.mapper.BrokerMessageLogMapper;**import** com.myimooc.rabbitmq.ha.dao.po.BrokerMessageLogPO;**import** org.springframework.amqp.rabbit.core.RabbitTemplate;**import** org.springframework.amqp.rabbit.support.CorrelationData;**import** org.springframework.beans.factory.annotation.Autowired;**import** org.springframework.stereotype.Component;

*/\*\**

*\* <br>*

*\* 标题: 订单消息发送者<br>*

*\* 描述: 订单消息发送者<br>*

*\* 时间: 2018/09/06<br>*

*\**

*\* @author zc*

*\*/***@Componentpublic** **class** **OrderSender** {

**@Autowired**

**private** RabbitTemplate rabbitTemplate;

**@Autowired**

**private** BrokerMessageLogMapper brokerMessageLogMapper;

*/\*\**

*\* 回调方法：confirm确认*

*\*/*

**private** **final** RabbitTemplate.ConfirmCallback confirmCallback = **new** RabbitTemplate.ConfirmCallback() {

**@Override**

**public** **void** **confirm**(CorrelationData correlationData, **boolean** ack, String cause) {

System.out.println("correlationData：" + correlationData);

String messageId = correlationData.getId();

**if** (ack) {

*// 如果confirm返回成功，则进行更新*

BrokerMessageLogPO messageLogPO = **new** BrokerMessageLogPO();

messageLogPO.setMessageId(messageId);

messageLogPO.setStatus(Constants.OrderSendStatus.SEND\_SUCCESS);

brokerMessageLogMapper.changeBrokerMessageLogStatus(messageLogPO);

} **else** {

*// 失败则进行具体的后续操作：重试或者补偿等*

System.out.println("异常处理...");

}

}

};

*/\*\**

*\* 发送订单*

*\**

*\* @param order 订单*

*\*/*

**public** **void** **send**(Order order) {

*// 设置回调方法*

**this**.rabbitTemplate.setConfirmCallback(confirmCallback);

*// 消息ID*

CorrelationData correlationData = **new** CorrelationData(order.getMessageId());

*// 发送消息*

**this**.rabbitTemplate.convertAndSend("order-exchange", "order.a", order, correlationData);

}

}

2.编写OrderService类

**package** com.myimooc.rabbitmq.ha.service;

**import** com.myimooc.rabbitmq.entity.Order;**import** com.myimooc.rabbitmq.ha.constant.Constants;**import** com.myimooc.rabbitmq.ha.dao.mapper.BrokerMessageLogMapper;**import** com.myimooc.rabbitmq.ha.dao.mapper.OrderMapper;**import** com.myimooc.rabbitmq.ha.dao.po.BrokerMessageLogPO;**import** com.myimooc.rabbitmq.ha.producer.OrderSender;**import** com.myimooc.rabbitmq.ha.util.FastJsonConvertUtils;**import** org.apache.commons.lang3.time.DateUtils;**import** org.springframework.beans.factory.annotation.Autowired;**import** org.springframework.stereotype.Service;

**import** java.util.Date;

*/\*\**

*\* <br>*

*\* 标题: 订单服务<br>*

*\* 描述: 订单服务<br>*

*\* 时间: 2018/09/07<br>*

*\**

*\* @author zc*

*\*/***@Servicepublic** **class** **OrderService** {

**@Autowired**

**private** OrderMapper orderMapper;

**@Autowired**

**private** BrokerMessageLogMapper brokerMessageLogMapper;

**@Autowired**

**private** OrderSender orderSender;

*/\*\**

*\* 创建订单*

*\**

*\* @param order 订单*

*\*/*

**public** **void** **create**(Order order) {

*// 当前时间*

Date orderTime = **new** Date();

*// 业务数据入库*

**this**.orderMapper.insert(order);

*// 消息日志入库*

BrokerMessageLogPO messageLogPO = **new** BrokerMessageLogPO();

messageLogPO.setMessageId(order.getMessageId());

messageLogPO.setMessage(FastJsonConvertUtils.convertObjectToJson(order));

messageLogPO.setTryCount(0);

messageLogPO.setStatus(Constants.OrderSendStatus.SENDING);

messageLogPO.setNextRetry(DateUtils.addMinutes(orderTime, Constants.ORDER\_TIMEOUT));

**this**.brokerMessageLogMapper.insert(messageLogPO);

*// 发送消息*

**this**.orderSender.send(order);

}

}

3.编写RetryMessageTask类

**package** com.myimooc.rabbitmq.ha.task;

**import** com.myimooc.rabbitmq.entity.Order;**import** com.myimooc.rabbitmq.ha.constant.Constants;**import** com.myimooc.rabbitmq.ha.dao.mapper.BrokerMessageLogMapper;**import** com.myimooc.rabbitmq.ha.dao.po.BrokerMessageLogPO;**import** com.myimooc.rabbitmq.ha.producer.OrderSender;**import** com.myimooc.rabbitmq.ha.util.FastJsonConvertUtils;**import** org.slf4j.Logger;**import** org.slf4j.LoggerFactory;**import** org.springframework.beans.factory.**annotation**.Autowired;**import** org.springframework.scheduling.**annotation**.Scheduled;**import** org.springframework.stereotype.Component;

**import** java.util.List;

*/\*\**

*\* <br>*

*\* 标题: 重发消息定时任务<br>*

*\* 描述: 重发消息定时任务<br>*

*\* 时间: 2018/09/07<br>*

*\**

*\* @author zc*

*\*/***@Componentpublic** **class** **RetryMessageTask** {

**private** Logger logger = LoggerFactory.getLogger(getClass());

**@Autowired**

**private** OrderSender orderSender;

**@Autowired**

**private** BrokerMessageLogMapper brokerMessageLogMapper;

*/\*\**

*\* 启动完成3秒后开始执行，每隔10秒执行一次*

*\*/*

**@Scheduled(initialDelay = 3000, fixedDelay = 10000)**

**public** void retrySend() {

logger.debug("重发消息定时任务开始");

*// 查询 status = 0 和 timeout 的消息日志*

List<BrokerMessageLogPO> pos = **this**.brokerMessageLogMapper.listSendFailureAndTimeoutMessage();

**for** (BrokerMessageLogPO po : pos) {

logger.debug("处理消息日志：{}",po);

**if** (po.getTryCount() >= Constants.MAX\_RETRY\_COUNT) {

*// 更新状态为失败*

BrokerMessageLogPO messageLogPO = new BrokerMessageLogPO();

messageLogPO.setMessageId(po.getMessageId());

messageLogPO.setStatus(Constants.OrderSendStatus.SEND\_FAILURE);

**this**.brokerMessageLogMapper.changeBrokerMessageLogStatus(messageLogPO);

} **else** {

*// 进行重试，重试次数+1*

**this**.brokerMessageLogMapper.updateRetryCount(po);

Order reSendOrder = FastJsonConvertUtils.convertJsonToObject(po.getMessage(), Order.**class**);

**try** {

**this**.orderSender.send(reSendOrder);

} **catch** (Exception ex) {

*// 异常处理*

logger.error("消息发送异常：{}", ex);

}

}

}

logger.debug("重发消息定时任务结束");

}

}

4.编写ApplicationTest类

package com.myimooc.rabbitmq.ha;

**import** com.myimooc.rabbitmq.entity.Order;**import** com.myimooc.rabbitmq.ha.service.OrderService;**import** org.junit.Test;**import** org.junit.runner.RunWith;**import** org.springframework.beans.factory.annotation.Autowired;**import** org.springframework.boot.test.context.SpringBootTest;**import** org.springframework.test.context.junit4.SpringRunner;

**import** java.util.UUID;

*/\*\**

*\* <br>*

*\* 标题: 订单创建测试<br>*

*\* 描述: 订单创建测试<br>*

*\* 时间: 2018/09/07<br>*

*\**

*\* @author zc*

*\*/*

@**RunWith**(**SpringRunner**.**class**)

@**SpringBootTestpublic** **class** **ApplicationTest** {

@**Autowired**

**private** **OrderService** orderService;

@**Test**

**public** void testCreateOrder(){

**Order** order = new **Order**();

order.setId(**String**.valueOf(**System**.currentTimeMillis()));

order.setName("测试创建订单");

order.setMessageId(**System**.currentTimeMillis() + "$" + **UUID**.randomUUID().toString().replaceAll("-",""));

this.orderService.create(order);

}

}

# RabbitMQ高级特性

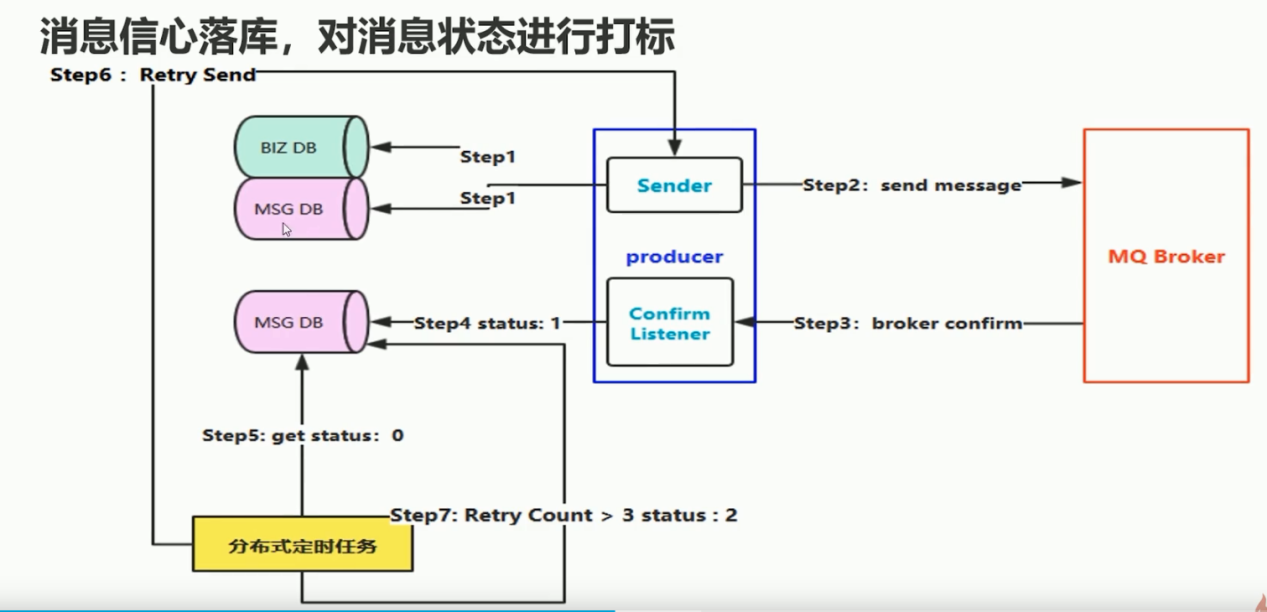
## 消息如何保障100%的投递成功？

### 什么是生产端的可靠性投递？

* 保障消息发出去
* 保障MQ节点的成功接收
* 发送端收到MQ节点（Broker）确认应答
* 完善的消息进行补偿机制

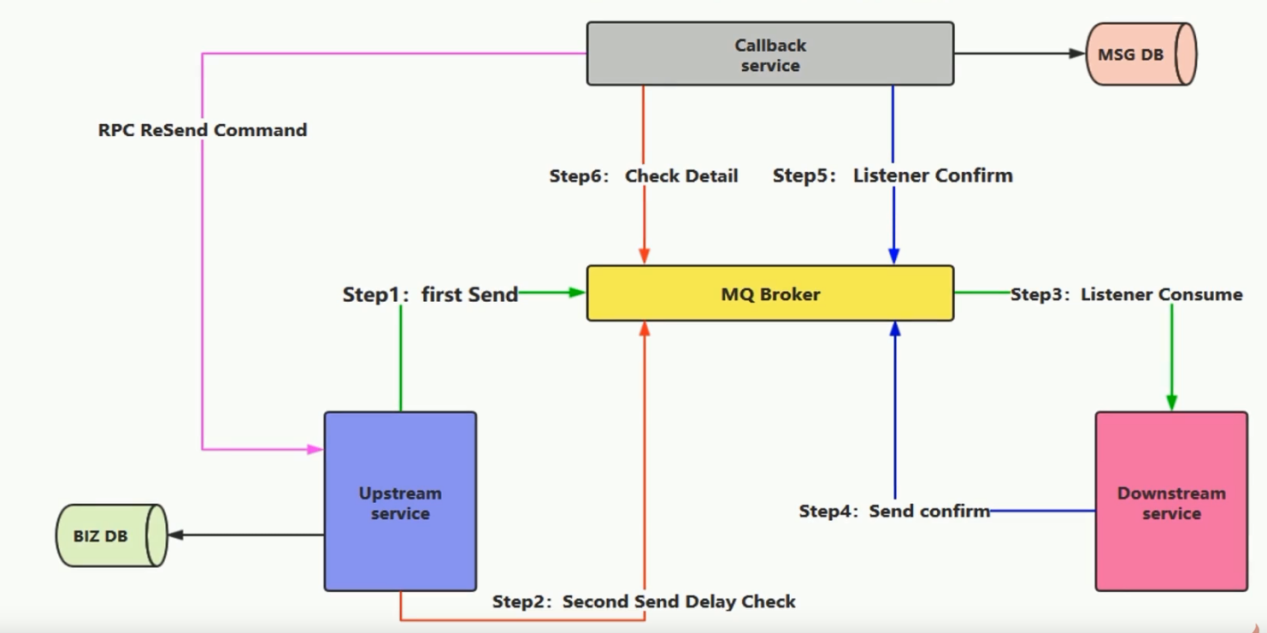
### BAT/TMD 互联网大厂的解决方案

* 消息落库，对消息状态进行打标



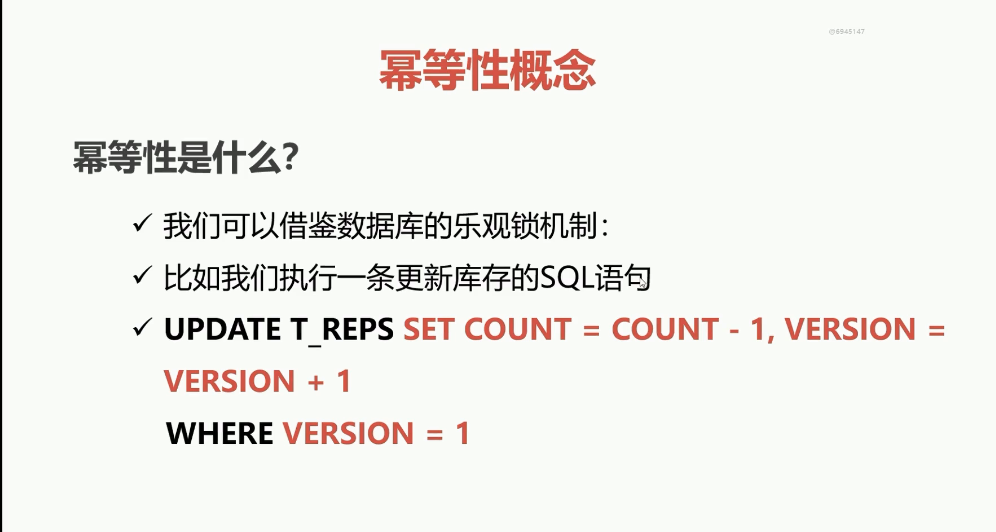
保障MQ我们思考如果第一种可靠性投递，在高并发的场景下是否适合？

* 消息的延迟投递，做二次检查，回调检查（节省数据入库的操作）



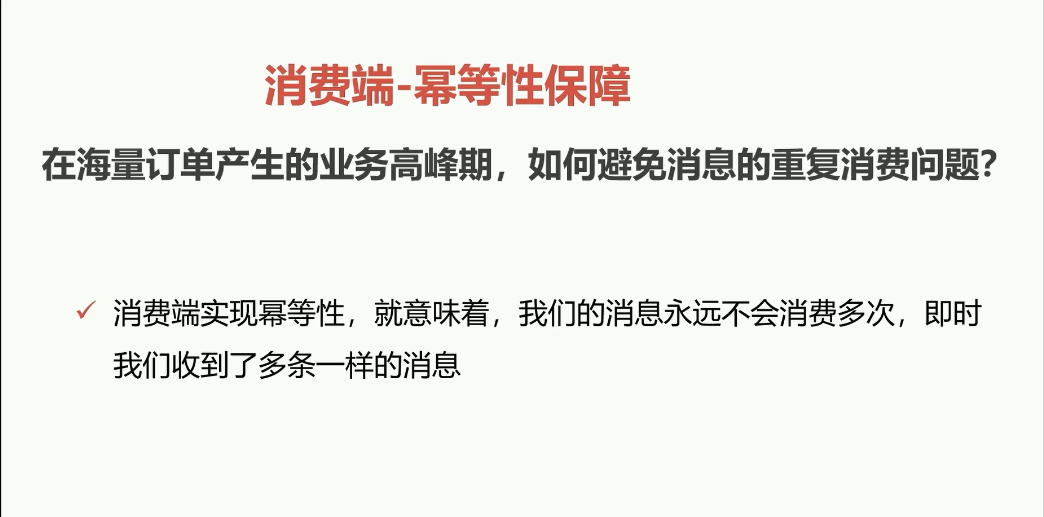
## 幂等性概念和业界主流解决方案

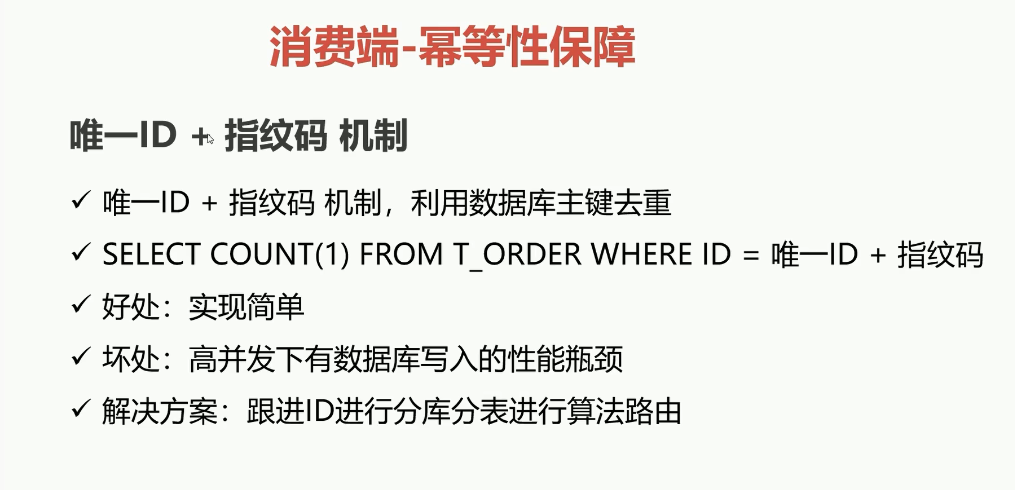
#### 幂等性概念

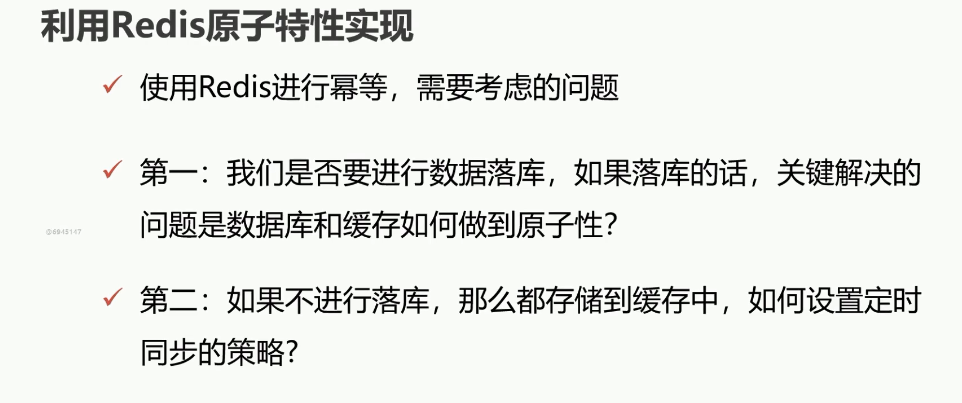


elasticsearch就是严格的幂等性操作

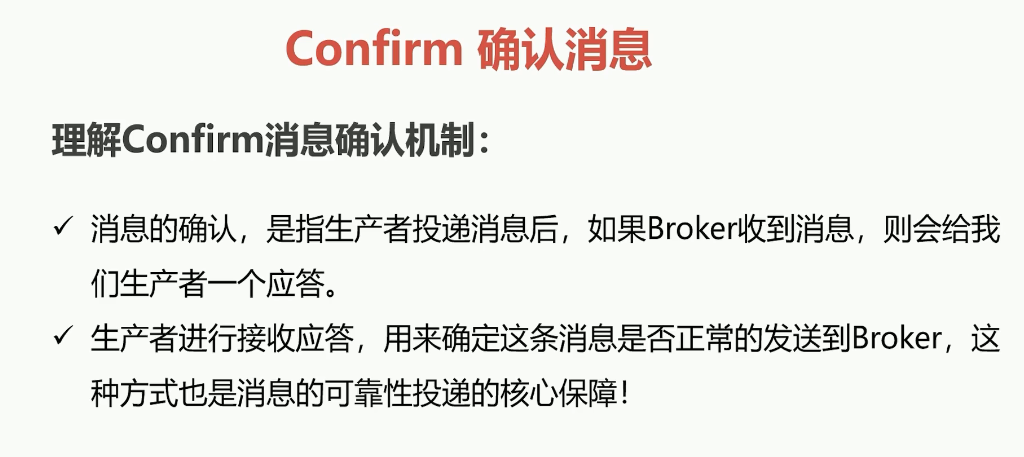
#### 消费端-幂等性保障

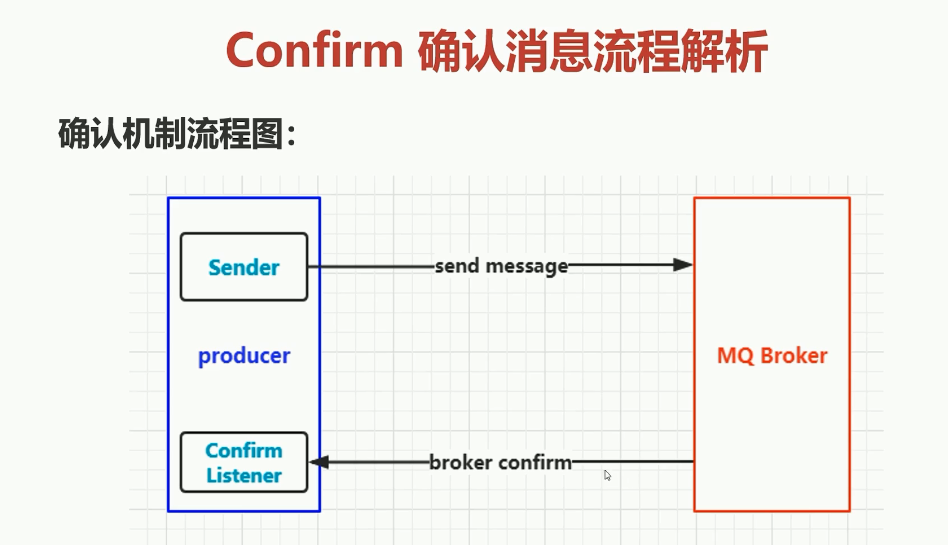


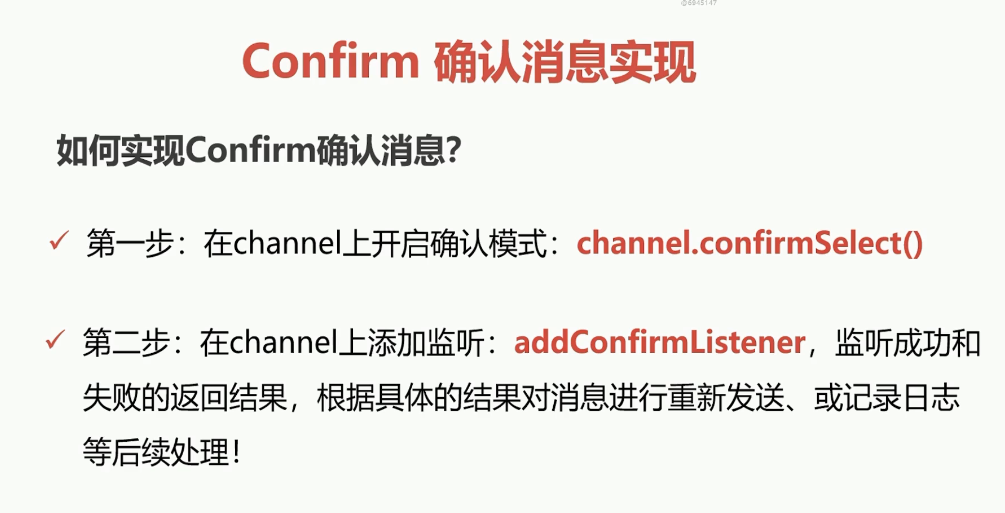




## Confirm确认消息详解





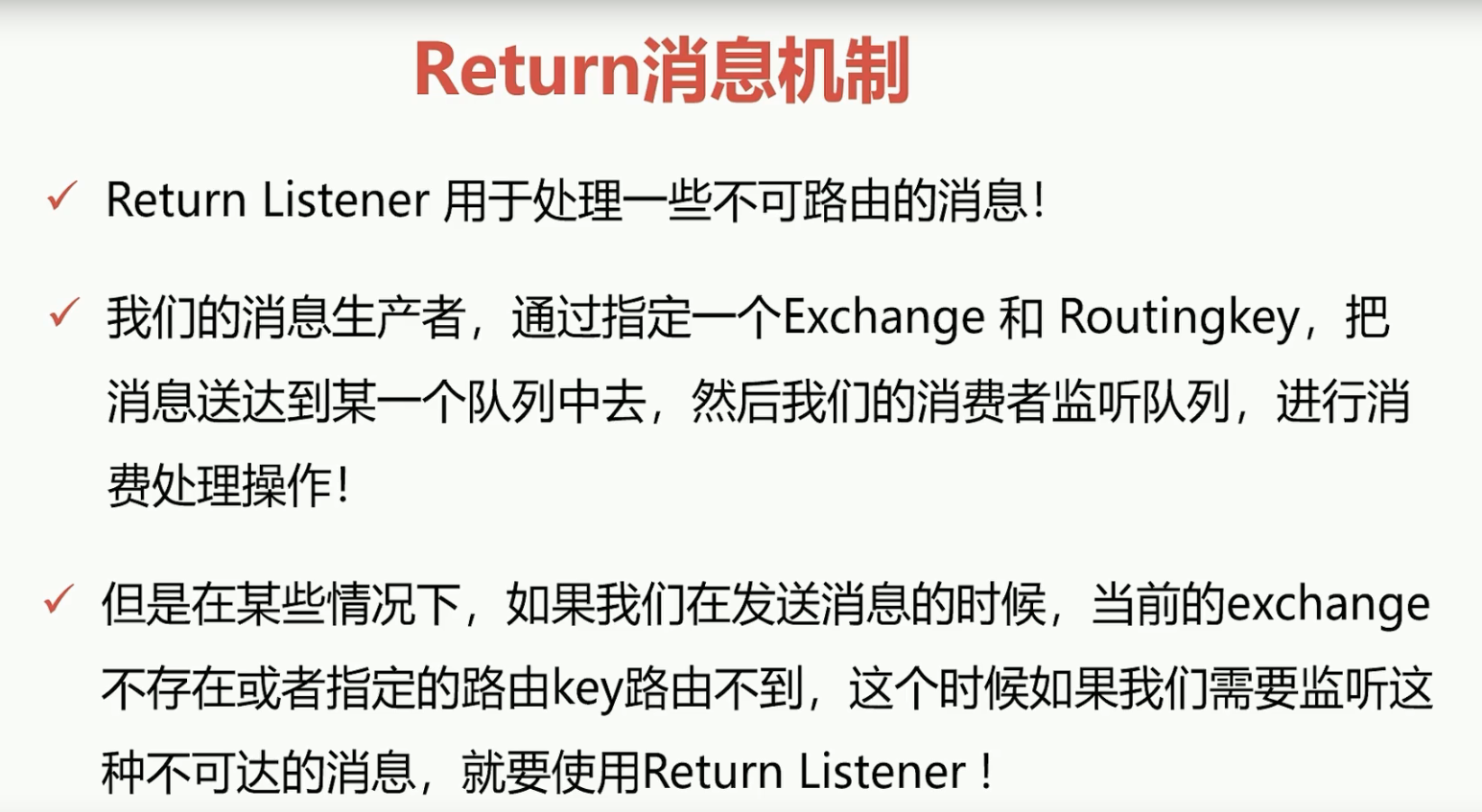


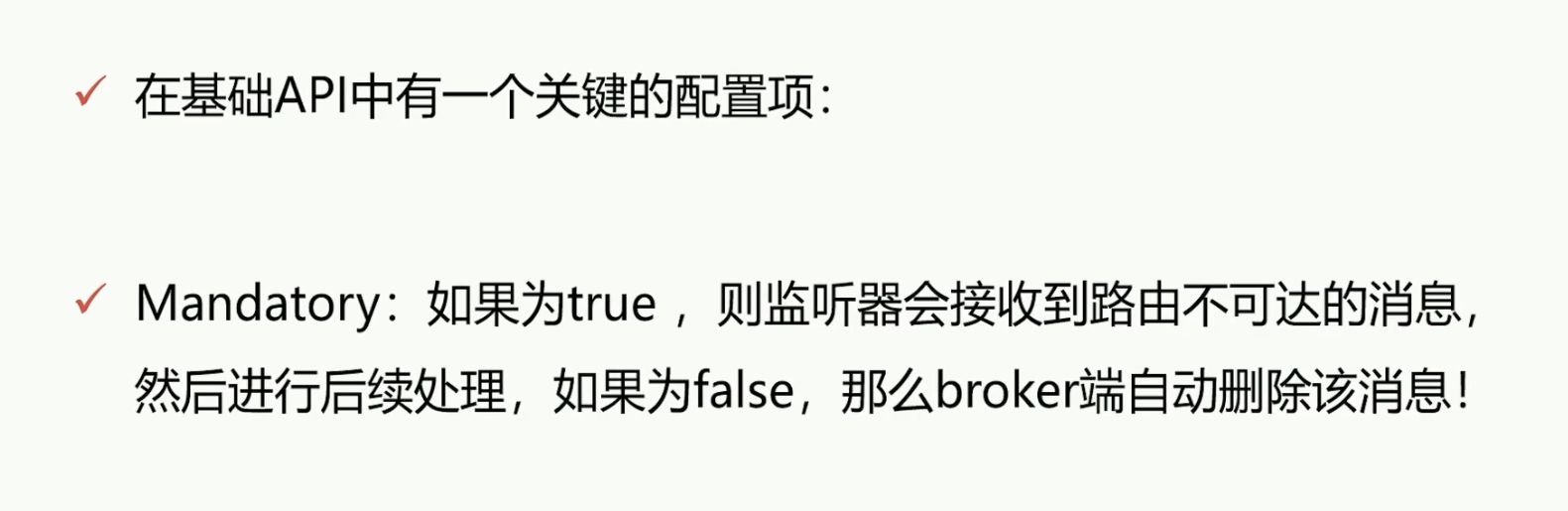
磁盘满了，队列满了都会导致返回nack

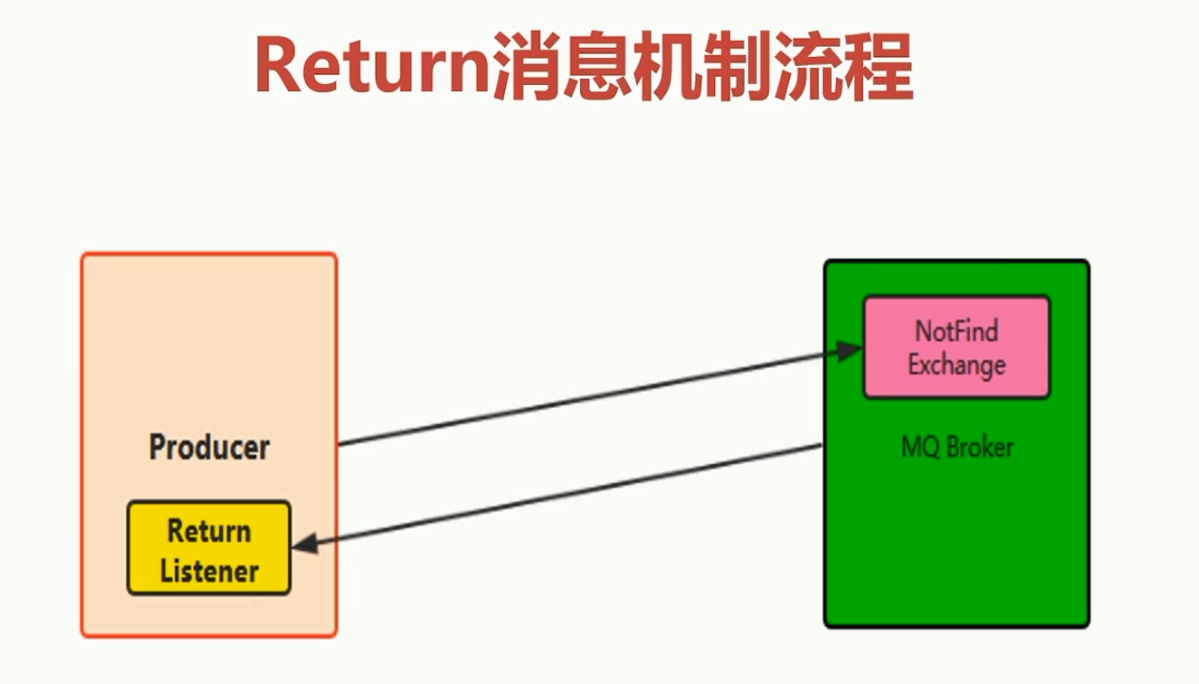
如果出现网络闪断，会出现既不返回ack也不返回nack

## Return 消息机制详解

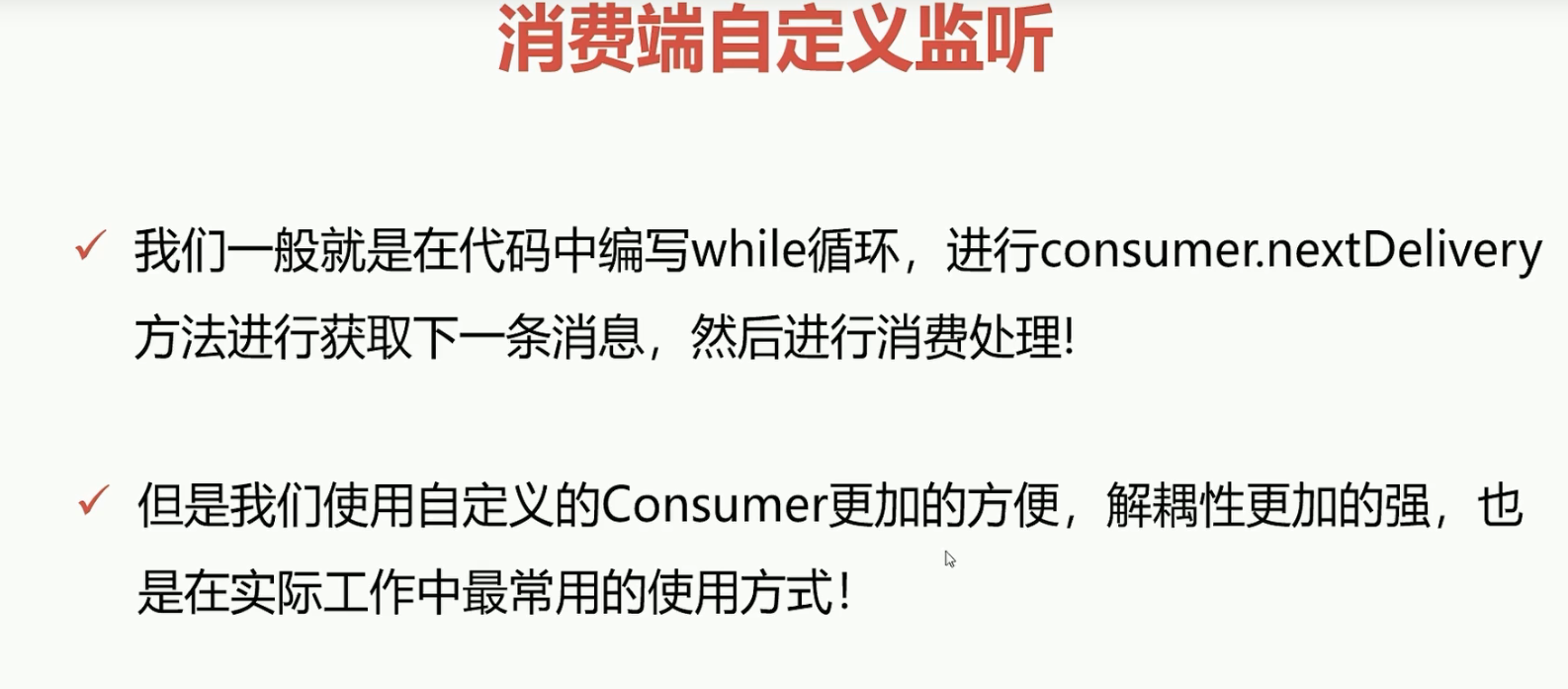
监听无用不可达的消息

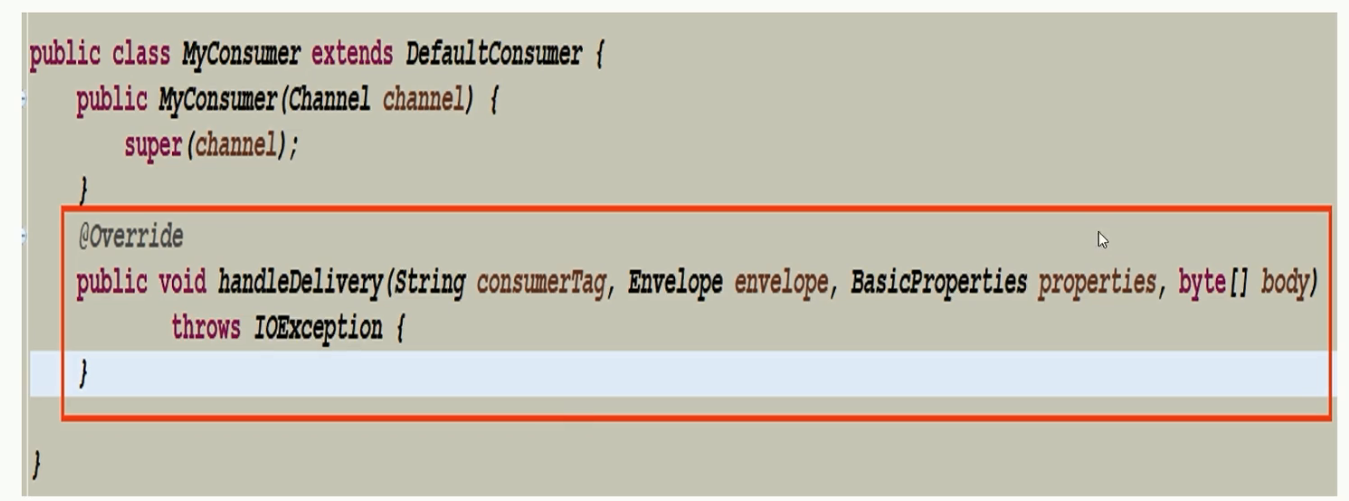




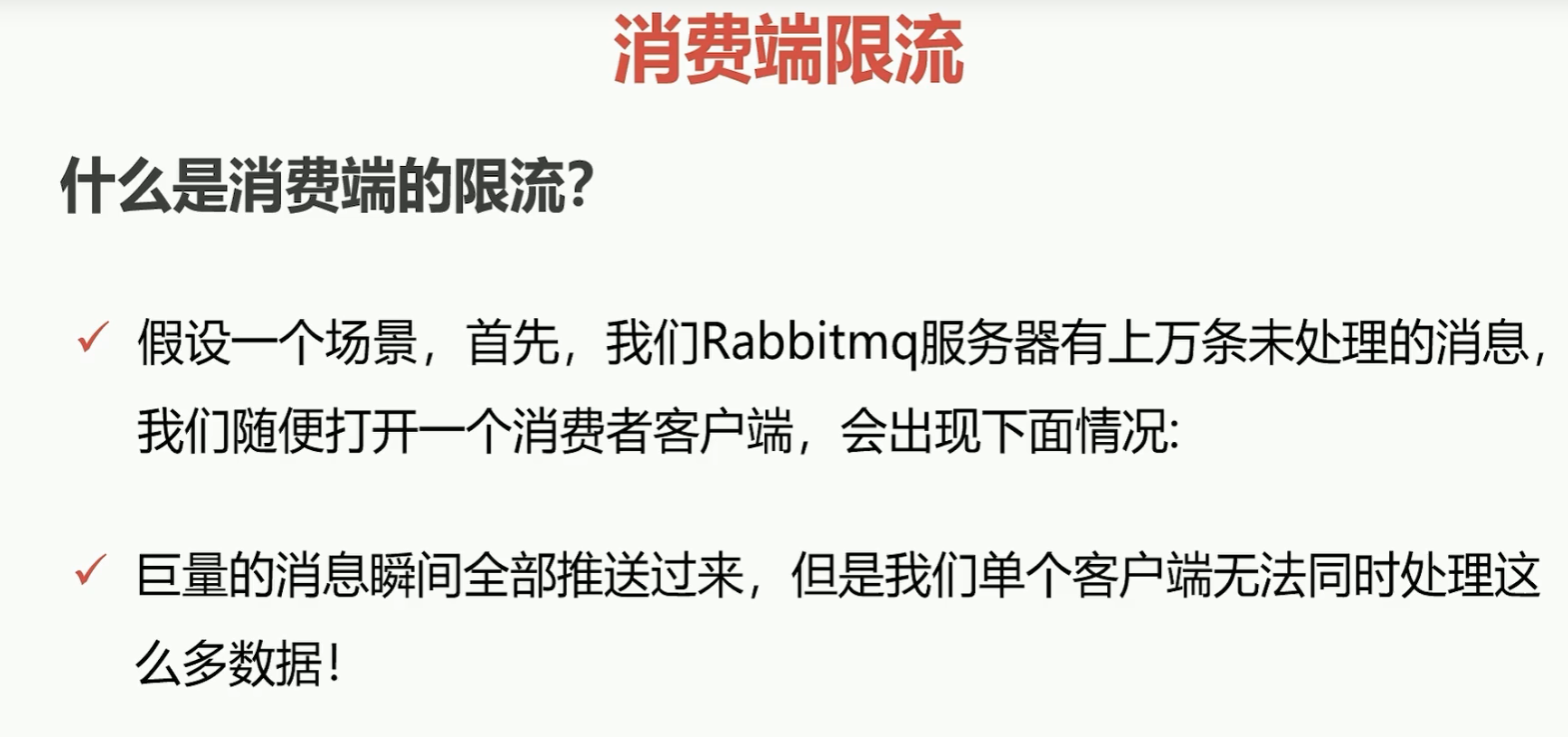


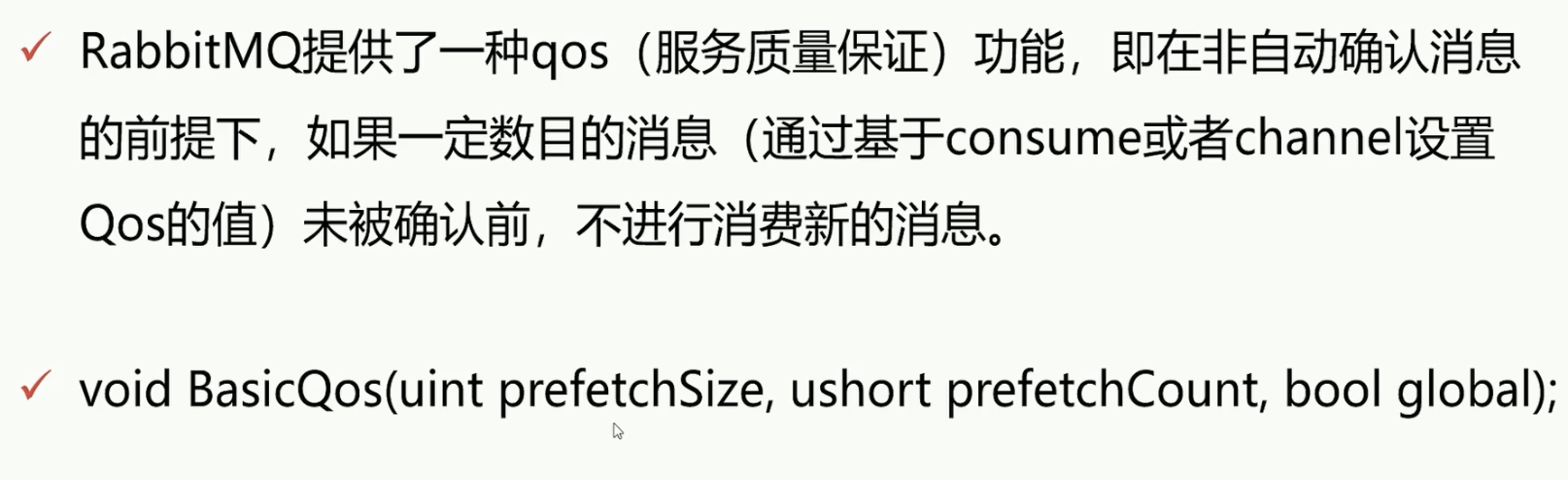
## 自定义消费端使用



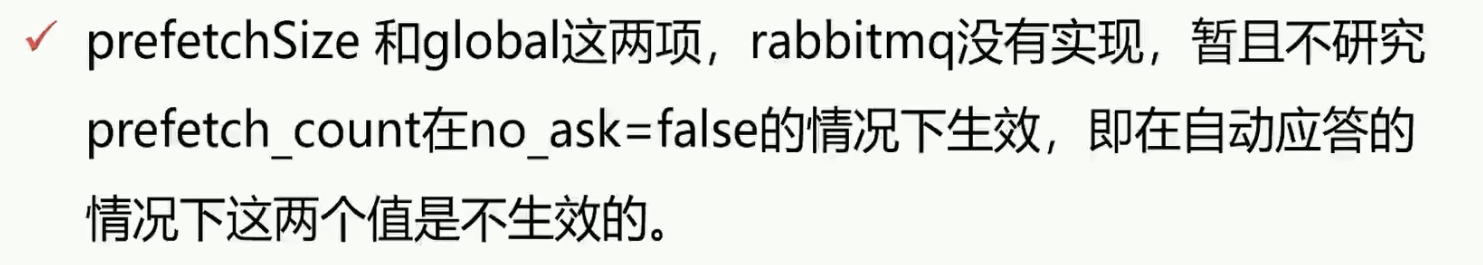


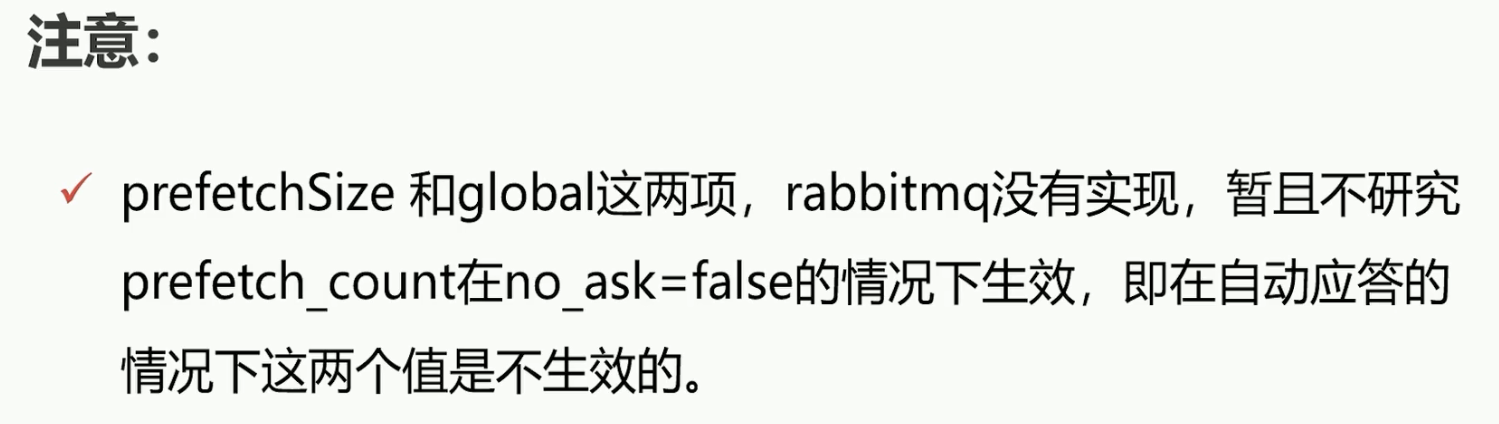
## 消费端的限流政策



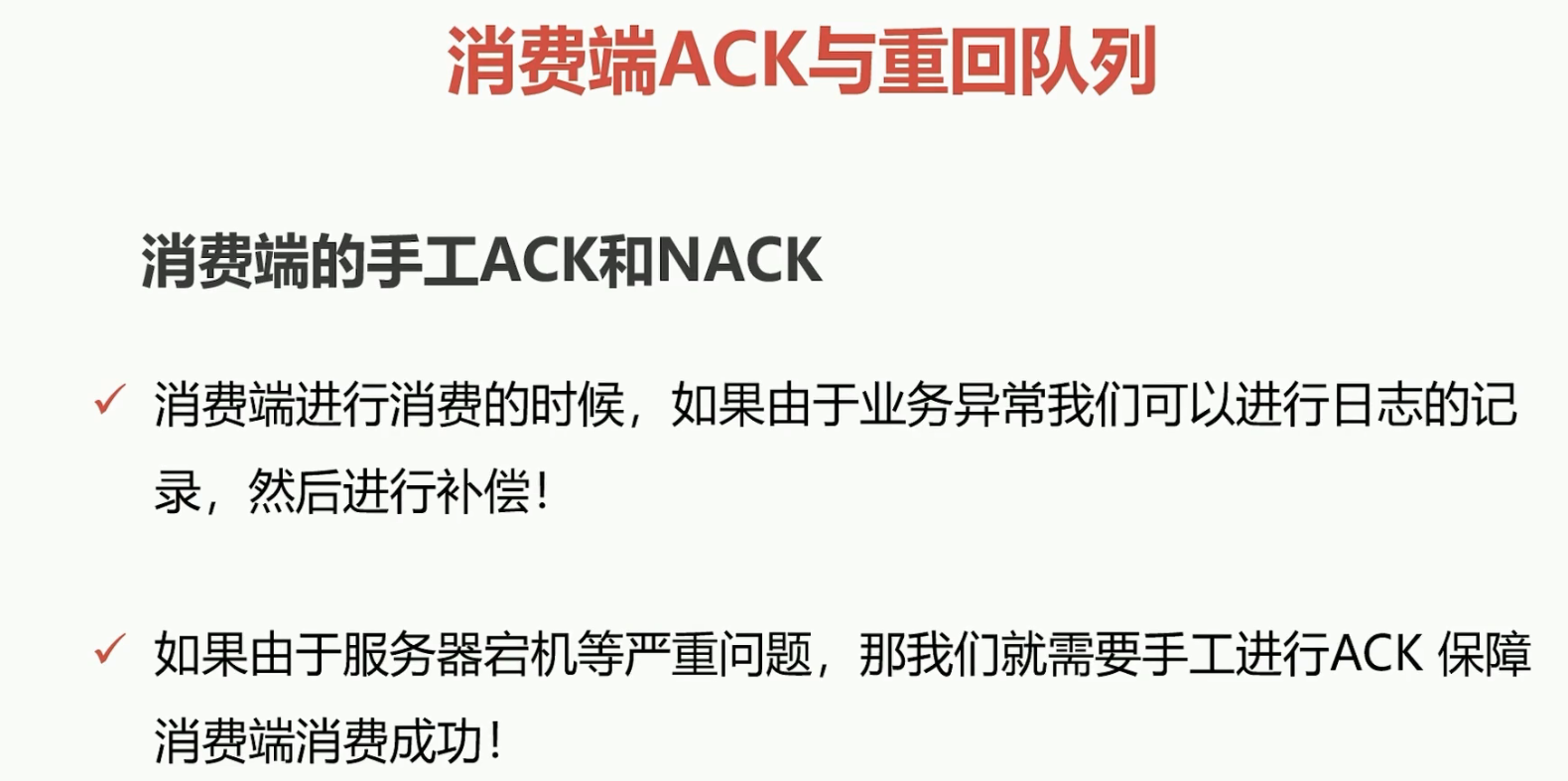


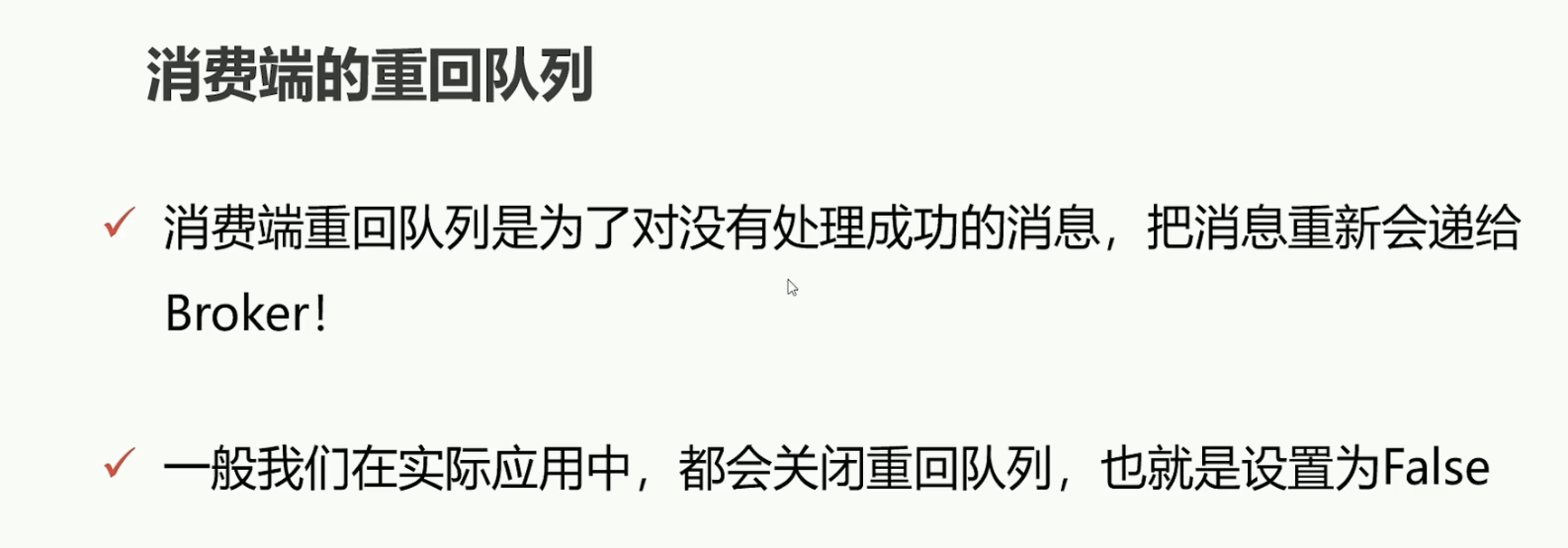




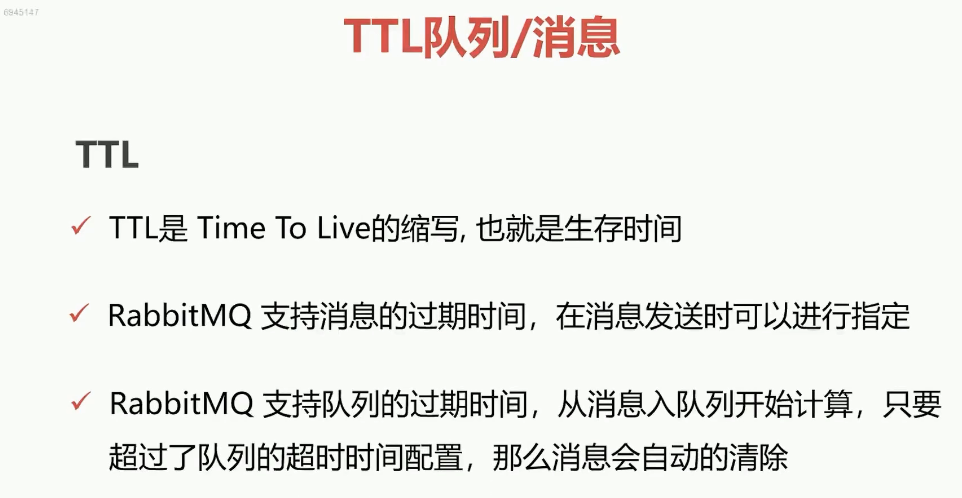


## 消费端的ack与重回队列机制





## TTL消息详解



## 死信队列详解及本章小结

