

重点：加二当家小D讲师微信：xdclass6

有专属技术交流群，有问题直接留言给我即可，扫描下面二维码也可以





今天谁也不能阻止我学习

 小滴课堂 愿景："让编程不再难学，让技术与生活更加有趣"

第一章 架构师成长-高性能消息队列 RabbitMQ课程介绍

第1集 架构成长-高性能消息队列RabbitMQ介绍

简介：讲解高性能消息队列RabbitMQ适合人员和学后水平

- 课程介绍

从0到1讲解高性能消息队列RabbitMQ，急速入门Docker部署RabbitMQ，掌握RabbitMQ核心概念和多种工作模式，不止讲解MQ多个工作模式，如简单队列、工作队列(轮训+公平策略)、发布订阅、路由模式、主题模式等，还整合当下新版热门框架SpringBoot2.X；

高级内容讲解消息可靠性投递、ACK消费确认模式，玩转死信队列，实现延迟队列功能+案例实战

搭建RabbitMQ普通集群、Mirror镜像集群，整合SpringBoot模拟多种异常情况

高级工程师+架构师必备大厂面试题+原理。

一套真正让你掌握RabbitMQ核心应用+内功的视频，总共15章近60集视频，全新版本录制



别嫌我啰嗦！

- 为什么要学习RabbitMQ消息队列
 - 多数互联网公司里面用的技术栈，可以承载海量消息处

理

- 在多数互联网公司中， RabbitMQ占有率很高，且全球都很流行
- 在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现强劲，与SpringAMQP完美的整合、API丰富易用
- 可以作为公司内部培训技术分享必备知识，可靠性投递、消费、高可用集群等

- 学后水平

- 零基础掌握MQ中间件应用场景、掌握多个业界主流中间件优缺点
- 掌握JMS、AMQP核心概念，各个优缺点和适合场景
- 零基础急速掌握Docker容器知识+Linux搭建Docker和部署RabbitMQ
- 掌握RabbitMQ多个核心概念交换机、队列、虚拟主机和Web管控台使用
- 玩转RabbitMQ多个工作队列、发布订阅模型、主题模型通配符实战
- 掌握新版SpringBoot+AMQP整合RabbitMQ并开发多个模式实战
- 高级篇幅玩转可靠性消息投递ConfirmCallback和returnCallback编码实战

- 高级篇幅掌握消息ACK确认机制+多种Reject编码实战
 - 高级篇幅掌握RabbitMQ TTL死信队列 + 延迟队列开发
【综合案例实战】
 - 高级加餐内容
 - 掌握Docker搭建RabbitMQ高可用 默认、Mirror镜像集群搭建实战+适合场景
 - 掌握多个集群模式整合SpringBoot，模拟多种异常场景配置实战
 - 掌握常见一线互联网大厂RabbitMQ面试题+核心原理+ 学习路线
-
- 适合人群
 - 高级后端工程师、高级前端/全栈工程师、运维工程师、CTO 更新必备技术栈
 - 从传统软件公司过渡到互联网公司的人员
 - 课程技术技术栈和环境说明
 - MQ版本： RabbitMQ3.8.9 + ErLang23.2.1
 - SpringBoot.2.4 + Maven + IDEA旗舰版 + JDK8 或 JDK11
 - 学习形式
 - 视频讲解 + 文字笔记 + 原理分析 + 交互流程图
 - 配套源码 + 笔记 + 专属技术群答疑(我答疑， 联系客服)

进群)

- 进技术群还有加餐内容（多种高可用集群搭建+大厂面试题），大厂内推资源、文档大礼包等
- 课程有配套的源码，在每章-每集的资料里面，如果没有用到代码就不保存
- 只要是我的课程和项目-我会一直维护下去，大家不用担心！！！

第2集 高性能消息队列RabbitMQ大纲速览和效果演示

简介：讲解高性能消息队列RabbitMQ课程大纲和效果演示

- 课程效果演示

我信你个鬼 你个糟老头子



- 课程学前基础
 - SpringBoot基础+IDEA基础
 - PS:不会上面的基础也没关系,这些基础都有课程, 【联系我们客服】即可, 且是刚录制的新版课程
- 目录大纲浏览
- 学习寄语
 - 保持谦虚好学、术业有专攻、在校的学生, 有些知识掌握也比工作好几年掌握的人厉害 课程有配套的源码, 在每章-每集的资料里面, 如果没用到代码就不保存。
 - 目录介绍如果自己操作的情况和视频不一样, 导入课程代码对比验证基本就可以发现问题了 官方要求, 务必保持版本一致, 不然会遇到很多问题。
 - 对于看不懂的视频, 一律归因于写或讲的人太蠢, 当然也不一定是事实, 但这样的思考方式能让我避免陷入自我怀疑的负面情绪

第3集 课程相关开发环境准备和新版 SpringBoot2.X项目创建

简介：课程开发环境准备和说明

- 必备基础环境：JDK8或者JDK11版本 + Maven3.5(采用默认) + IDEA旗舰版 + Mysql5.7以上版本
 - 不要用JDK11以上，非大规模的LTS版本且多数软件不支持
 - 2021~2024年内，JDK11会是大规模流行
- 操作系统：Win10或者Mac苹果
- 创建新版SpringBoot2.X项目
 - <https://spring.io/projects/spring-boot>
 - 在线构建工具 <https://start.spring.io/>

- 如果版本或者链接找不到， 可以直接导入课程项目
- 注意: 有些包maven下载慢， 等待下载如果失败
 - 删除本地仓库spring相关的包， 重新执行 mvn install
 - 建议先使用默认的maven仓库， 不用更换地址
- 当前项目仓库地址修改

```
<!-- 代码库 -->
<repositories>
    <repository>
        <id>maven-ali</id>

        <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
</repositories>
```

```
<pluginRepositories>
    <pluginRepository>
        <id>public</id>
        <name>aliyun nexus</name>

        <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </pluginRepository>
</pluginRepositories>
```



小滴课堂 愿景："让编程不再难学，让技术与生活更加有趣"

第二章 大话MQ消息中间件 +JMS+AMQP核心知识

第1集 什么是MQ消息中间件和应用场景

简介：介绍什么是MQ消息中间件和应用场景

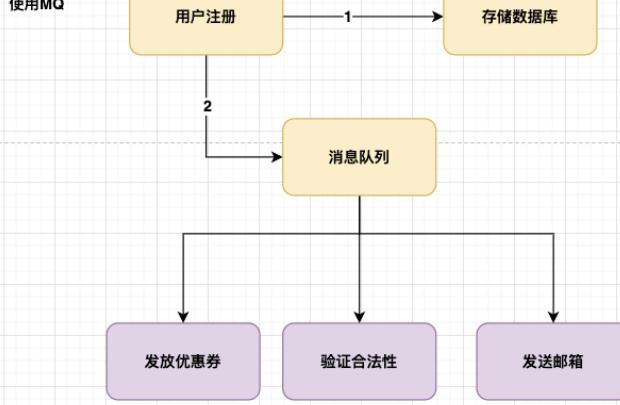
- 什么是MQ消息中间件
 - 全称MessageQueue，主要是用于程序和程序直接通信，异步+解耦
- 使用场景：
 - 核心应用
 - 解耦：订单系统-> 物流系统
 - 异步：用户注册-> 发送邮件，初始化信息
 - 削峰：秒杀、日志处理
 - 跨平台、多语言
 - 分布式事务、最终一致性
 - RPC调用上下游对接，数据源变动->通知下属

队列主要作用：解耦，异步和并行，用户注册的案例来说明MQ的作用



小滴课堂-架构师系列之玩转高性能消息队列专题

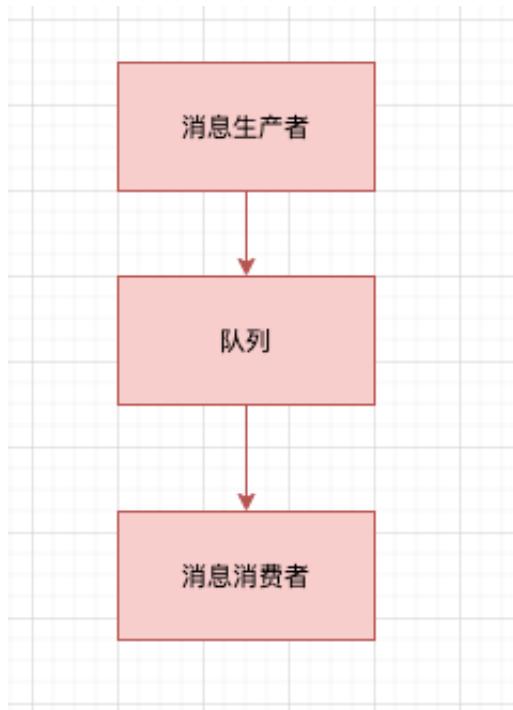
使用MQ



第2集 JMS消息服务和常见核心概念介绍

简介：讲解什么是AMQP和JMS消息服务

- 什么是JMS: Java消息服务 (Java Message Service),Java平台中关于面向消息中间件的接口
 - JMS是一种与厂商无关的 API，用来访问消息收发系统消息，它类似于JDBC(Java Database Connectivity)。这里， JDBC 是可以用来访问许多不同关系数据库的 API
 - 是由Sun公司早期提出的消息标准，旨在为java应用提供统一的消息操作，包括create、 send、 receive
 - JMS是针对java的，那微软开发了NMS (.NET消息传递服务)



- 特性
 - 面向Java平台的标准消息传递API

- 在Java或JVM语言比如Scala、Groovy中具有互用性
- 无需担心底层协议
- 有queues和topics两种消息传递模型
- 支持事务、能够定义消息格式（消息头、属性和内容）
- 常见概念
 - JMS提供者：连接面向消息中间件的，JMS接口的一个实现，RocketMQ, ActiveMQ, Kafka等等
 - JMS生产者(Message Producer)：生产消息的服务
 - JMS消费者(Message Consumer)：消费消息的服务
 - JMS消息：数据对象
 - JMS队列：存储待消费消息的区域
 - JMS主题：一种支持发送消息给多个订阅者的机制
 - JMS消息通常有两种类型：点对点（Point-to-Point）、发布/订阅（Publish/Subscribe）



- 基础编程模型
 - MQ中需要用的一些类
 - ConnectionFactory：连接工厂，JMS用它创建连接
 - Connection：JMS客户端到JMS Provider的连接
 - Session：一个发送或接收消息的线程

- Destination : 消息的目的地;消息发送给谁.
- MessageConsumer / MessageProducer: 消息消费者，消息生产者

第3集 高级消息队列协议AMQP介绍和MQTT拓展

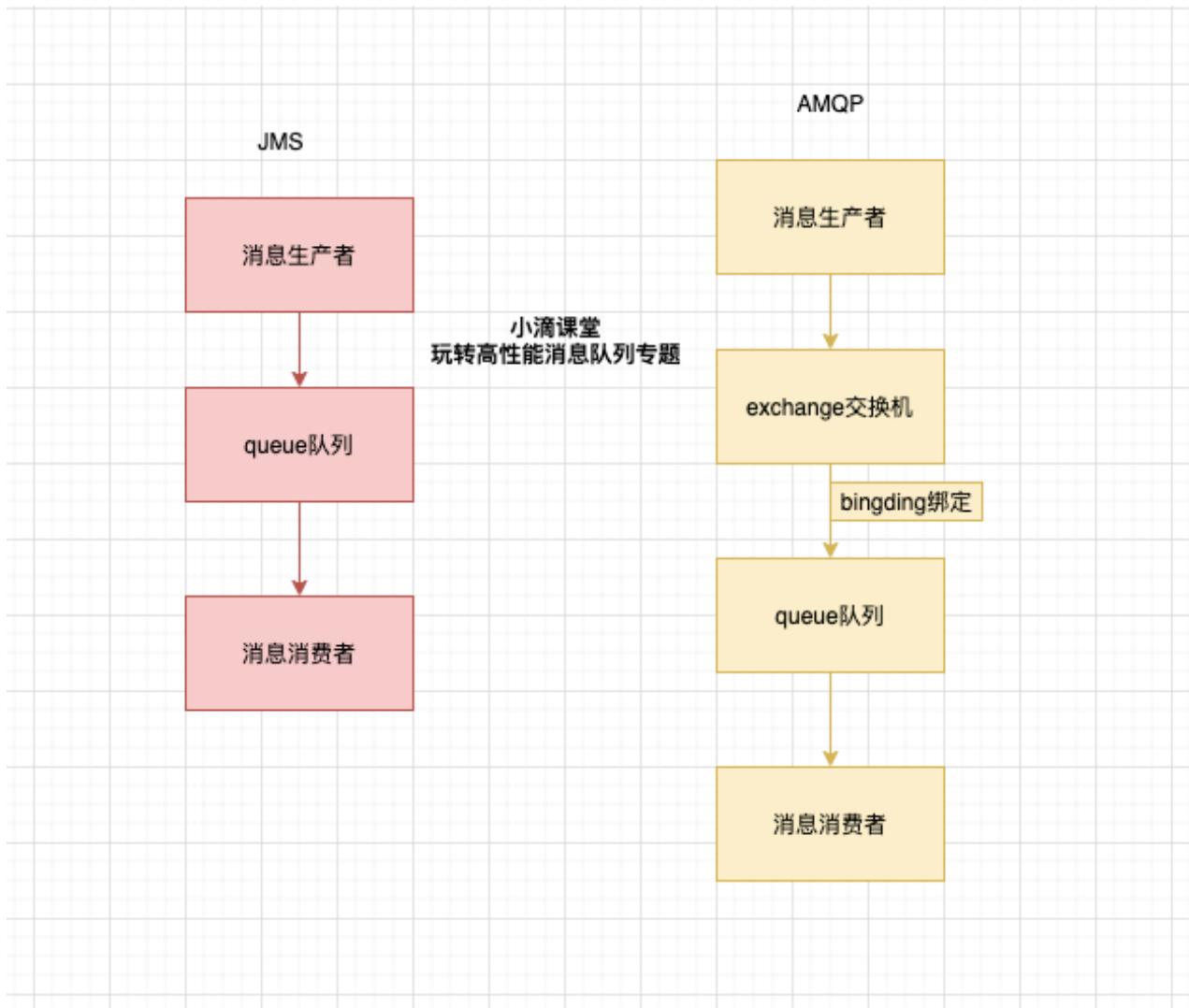
简介：介绍什么是AMQP高级消息队列协议和MQTT科普

- 背景
 - JMS或者NMS都没有标准的底层协议，它们可以在任何底层协议上运行，但是API是与编程语言绑定的，AMQP解决了这个问题，它使用了一套标准的底层协议



稳住 我们能赢!

- 什么是AMQP
 - AMQP (advanced message queuing protocol) 在2003年时被提出，最早用于解决金融领不同平台之间的消息传递交互问题,就是是一种协议，兼容JMS
 - 更准确说的链接协议 binary-wire-level-protocol 直接定义网络交换的数据格式，类似http
 - 具体的产品实现比较多，RabbitMQ就是其中一种



- 特性

- 独立于平台的底层消息传递协议
- 消费者驱动消息传递
- 跨语言和平台的互用性、属于底层协议
- 有5种交换类型direct, fanout, topic, headers, system
- 面向缓存的、可实现高性能、支持经典的消息队列，循环，存储和转发
- 支持长周期消息传递、支持事务（跨消息队列）

- AMQP和JMS的主要区别
 - AMQP不从API层进行限定，直接定义网络交换的数据格式,这使得实现了AMQP的provider天然性就是跨平台
 - 比如Java语言产生的消息，可以用其他语言比如python的进行消费
 - AMQP可以用http来进行类比，不关心实现接口的语言，只要都按照相应的数据格式去发送报文请求，不同语言的client可以和不同语言的server进行通讯
 - JMS消息类型：
TextMessage/ObjectMessage/StreamMessage等
 - AMQP消息类型：Byte[]
- 科普：大家可能也听过MQTT
 - MQTT: 消息队列遥测传输 (Message Queueing Telemetry Transport)
 - 背景：
 - 我们有面向基于Java的企业应用的JMS和面向所有其他应用需求的AMQP，那这个MQTT是做啥的？
 - 原因
 - 计算性能不高的设备不能适应AMQP上的复杂操作,MQTT它是专门为小设备设计的
 - MQTT主要是物联网 (IOT) 中大量的使用

- 特性

- 内存占用低，为小型无声设备之间通过低带宽发送短消息而设计
- 不支持长周期存储和转发，不允许分段消息（很难发送长消息）
- 支持主题发布-订阅、不支持事务（仅基本确认）
- 消息实际上是短暂的（短周期）
- 简单用户名和密码、不支持安全连接、消息不透明

第4集 架构师的解决方案-业界主流消息队列和技术选型讲解

简介：对比当下主流的消息队列和选择问题

- 业界主流的消息队列：Apache ActiveMQ、Kafka、RabbitMQ、RocketMQ

- ActiveMQ: <http://activemq.apache.org/>
 - Apache出品，历史悠久，支持多种语言的客户端和协议，支持多种语言Java, .NET, C++ 等
 - 基于JMS Provider的实现
 - 缺点：吞吐量不高，多队列的时候性能下降，存在消息丢失的情况，比较少大规模使用
- Kafka: <http://kafka.apache.org/>
 - 是由Apache软件基金会开发的一个开源流处理平台，由Scala和Java编写。Kafka是一种高吞吐量的分布式发布订阅消息系统，它可以处理大规模的网站中的所有动作流数据(网页浏览，搜索和其他用户的行动)，副本集机制，实现数据冗余，保障数据尽量不丢失；支持多个生产者和消费者
 - 类似MQ，功能较为简单，主要支持简单的MQ功能
 - 缺点：不支持批量和广播消息，运维难度大，文档比较少，需要掌握Scala
- RocketMQ: <http://rocketmq.apache.org/>
 - 阿里开源的一款的消息中间件，纯Java开发，具有高吞吐量、高可用性、适合大规模分布式系统应用的特点，性能强劲(零拷贝技术)，支持海量堆积，支持指定次数和时间间隔的失败消息重发，支持consumer端tag过滤、延迟消息等，在阿里内部进行大规模使用，适合在电商，互联网金融等领域
 - 基于JMS Provider的实现
 - 缺点：社区相对不活跃，更新比较快，纯java支持

- RabbitMQ: <http://www.rabbitmq.com/>
 - 是一个开源的AMQP实现，服务器端用Erlang语言编写，支持多种客户端，如：Python、Ruby、.NET、Java、C、用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不错
 - 缺点：使用Erlang开发，阅读和修改源码难度大
- 什么我们讲RabbitMQ呢，只要你目标是高级工程师或者架构师，就要多学，才知道解决方案+适合场景
 - 因为这个是rabbitmq专题课程
 - 下集专门介绍rabbitmq

我是一个开不起玩笑的人



**如果你和我开玩笑
我就当真**



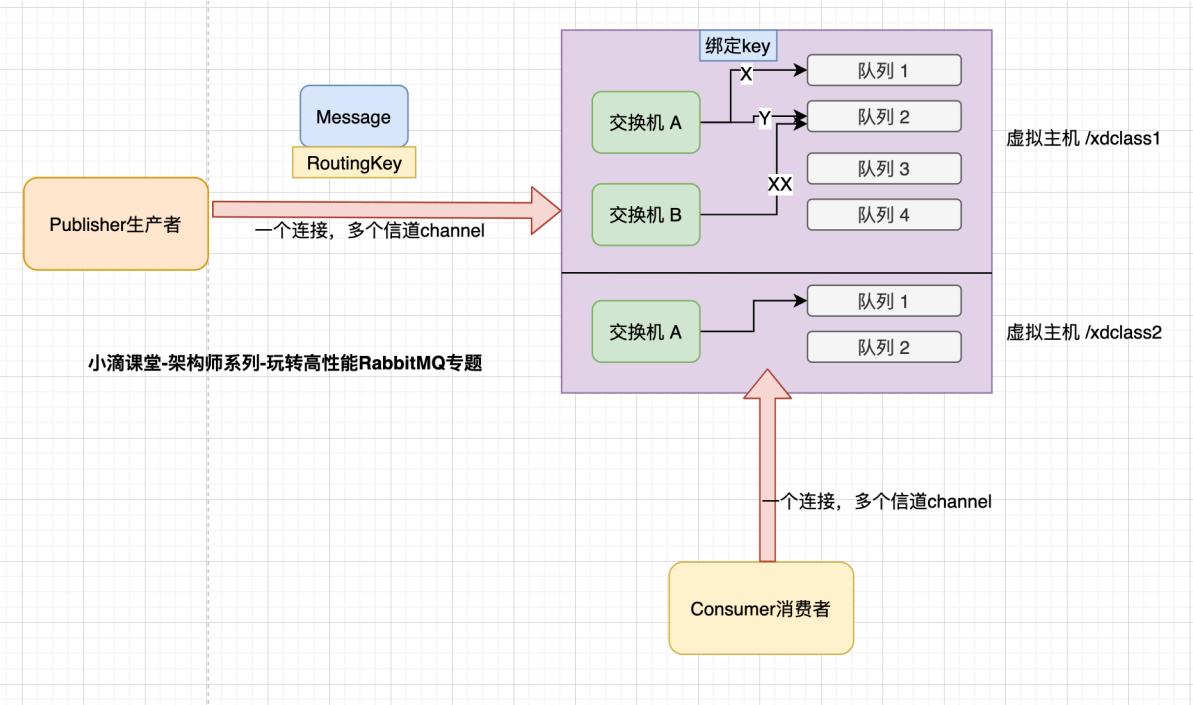
小滴课堂 愿景："让编程不再难学，让技术与生活更加有趣"

第三章 急速入门新版RabbitMQ核心概念+环境说明

第1集 什么是RabbitMQ消息队列和核心概念介绍

简介：介绍RabbitMQ消息队列

- RabbitMQ：<http://www.rabbitmq.com/>
 - 是一个开源的AMQP实现，服务器端用Erlang语言编写，支持多种客户端，如：Python、Ruby、.NET、Java、C，用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不错，与SpringAMQP完美的整合、API丰富易用
 - 文档：<https://www.rabbitmq.com/getstarted.html>
- 核心概念，了解了这些概念，是使用好RabbitMQ的基础



- Broker
 - RabbitMQ的服务端程序，可以认为一个mq节点就是一个broker
- Producer生产者
 - 创建消息Message，然后发布到RabbitMQ中
- Consumer消费者：
 - 消费队列里面的消息
- Message 消息
 - 生产消费的内容，有消息头和消息体，也包括多个属性配置，比如routingKey路由键
- Queue 队列
 - 是RabbitMQ 的内部对象，用于存储消息，消息都只能存储在队列中
- Channel 信道

- 一条支持多路复用的通道，独立的双向数据流通道，可以发布、订阅、接收消息。
 - 信道是建立在真实的TCP连接内的虚拟连接，复用TCP连接的通道
- Connection连接
 - 是RabbitMQ的socket链接，它封装了socket协议相关部分逻辑，一个连接上可以有多个channel进行通信
 - Exchange 交换器
 - 生产者将消息发送到 Exchange，交换器将消息路由到一个或者多个队列中，里面有多个类型，后续再一一介绍，队列和交换机是多对多的关系。
 - RoutingKey 路由键
 - 生产者将消息发给交换器的时候，一般会指定一个 RoutingKey，用来指定这个消息的路由规则
 - 最大长度255 字节
 - Binding 绑定
 - 通过绑定将交换器与队列关联起来，在绑定的时候一般会指定一个绑定键 (BindingKey)，这样 RabbitMQ 就知道如何正确地将消息路由到队列了

生产者将消息发送给交换器时，需要一个RoutingKey，当 BindingKey 和 RoutingKey 相匹配时，消息会被路由到对应的队列中

- Virtual host 虚拟主机

- 用于不同业务模块的逻辑隔离，一个Virtual Host里面可以有若干个Exchange和Queue，同一个VirtualHost 里面不能有相同名称的Exchange或Queue
- 默认是 /
 - /dev
 - /test
 - /pro

第2集 安装RabbitMQ 的Linux服务器环境准备说明

简介：安装RabbitMQ相关Linux服务器

- RabbitMQ安装方式
 - 安装文档 <https://www.rabbitmq.com/download.html>
 - 源码安装
 - 依赖多、且版本和维护相对复杂
 - 需要erlang环境、版本也是有要求
 - docker镜像安装【推荐】
 - 不用安装其他相关依赖，容器化部署是趋势
 - 方便管理维护，企业多采用这种方式
- Linux服务器准备：CentOS7.x以上即可, 根据个人能力选择哪种
 - 本地虚拟机
 - 不同系统容易出现确实依赖库，硬件也存在不兼容
 - 阿里云ECS服务器 【推荐，2核4g】
 - 统一环境，公司多基本都是用云服务器，对于工作上帮助大
- 本地开发
 - IDEA旗舰版 + JDK8 / JDK11 + Maven3.5以上 +

SpringBoot2.X

- 总结

- 接下来的第四章，如果大家对阿里云ECS和Docker有了了解，可以跳过这章
- 本章接下来第四章主要是讲阿里云服务器知识和Docker基础知识

第四章 阿里云Linux CentOS服务配置 +Docker核心急速入门

第1集 云服务器介绍和阿里云服务器ECS服务器选购

简介：什么是云服务器及目前主要的几个厂商介绍

- 云厂商
 - 阿里云：<https://www.aliyun.com/>
 - 腾讯云：<https://cloud.tencent.com/>
 - 亚马逊云：<https://aws.amazon.com/>
 - 阿里云新用户地址（如果地址失效，联系我或者客服即可，1折）
 - https://www.aliyun.com/minisite/goods?userCode=r5saexap&share_source=copy_link
- 环境问题说明
 - Win7、Win8、Win10、Mac、虚拟机等等，可能存在兼容问题
 - 务必使用CentOS 7 以上版本，64位系统！！！！
 - 选购实操

第2集 阿里云服务器远程登录和常用工具

简介：讲解阿里云服务器登录使用和常见终端工具

- 备注：(服务器、域名等使用你们自己购买的哈，上面有提供低价购买链接，失效找我)
- 阿里云新用户地址 https://www.aliyun.com/minisite/goods?userCode=r5saexap&share_source=copy_link
- 控制台修改阿里云远程连接密码
- windows工具 putty, xshell, security CRT
 - 参考资料：
 - <https://jingyan.baidu.com/article/e75057f210c6dcebc91a89dd.html>

- <https://www.jb51.net/softjc/88235.html>
- 苹果系统MAC： 通过终端登录
 - ssh root@ip 回车后输入密码
 - ssh root@120.24.216.117
- linux图形操作工具（用于远程连接上传文件）
 - mac: filezilla
 - sftp://120.24.216.117
 - windows: winscp
 - 参考资料：<https://jingyan.baidu.com/article/ed2a5d1f346fd409f6be179a.html>
- 更多阿里云操作，可以尝试自己通过百度进行找文档，安装mysql jdk nginx maven git redis等，也可以看我们的课程

第3集 分布式架构-容器化趋势Docker介绍和使用场景

简介： Docker介绍和使用场景

- 官网：<https://www.docker.com/get-started>
- 什么是Dokcer
 - 百科:一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。
 - 容器是完全使用沙箱机制，相互之间不会有任何接口，使用go语言编写，在LCX (linux容器) 基础上进行的封装
 - 简单来说：
 - 就是可以快速部署启动应用
 - 实现虚拟化，完整资源隔离
 - 一次编写，四处运行
 - 但有一定的限制，比如Docker是基于Linux 64bit 的，无法在32bit的linux/Windows/unix环境下使用
- 为什么要用
 - 提供一次性的环境，假如需要安装Mysql、RocketMQ、RabbitMQ，则需要安装很多依赖库、版本等，如果使用Docker则通过镜像就可以直接启动运行
 - 快速动态扩容，使用docker部署了一个应用，可以制作

成镜像，然后通过Docker快速启动

- 组建微服务架构，可以在一个机器上模拟出多个微服务，启动多个应用
 - 更好的资源隔离和共享
 - 一句话：开箱即用，快速部署，可移植性强，环境隔离
- 大课训练营里面的应用，PaaS云平台，容器编排调度，自动化扩容

第4集 阿里云Linux云服务器Centos 64位安装Docker实战

简介：讲解阿里云ECS服务安装Docker实战

- 远程连接ECS实例 8.129.113.233
- 依次运行以下命令添加yum源。

```
yum update
```

```
yum install epel-release -y
```

```
yum clean all
```

```
yum list
```

- 安装并运行Docker。

```
yum install docker-io -y
```

```
systemctl start docker
```

- 检查安装结果。

```
docker info
```

- 启动使用Docker

```
systemctl start docker      #运行Docker守护进程
```

```
systemctl stop docker      #停止Docker守护进程
```

```
systemctl restart docker   #重启Docker守护进程
```

- 更多文档

```
https://help.aliyun.com/document\_detail/51853.html?spm=a2c4g.11186623.6.820.RaToNY
```

第5集 面向对象的方式快速掌握 Docker仓库、镜像、容器核心概念

简介：快速掌握Dokcer基础知识

- 概念：

- Docker 镜像 - Docker images：容器运行的只读模板，操作系统+软件运行环境+用户程序

```
class User{  
    private String userName;  
    private int age;  
}
```

- Docker 容器 - Docker containers：容器包含了某个应用运行所需要的全部环境

```
User user = new User()
```

- Docker 仓库 - Docker registries：用来保存镜像，有公有和私有仓库，好比Maven的中央仓库和本地私服
- 总结 对比面向对象的方式

Dokcer 里面的镜像 : Java里面的类 Class
Docker 里面的容器 : Java里面的对象 Object
通过类创建对象，通过镜像创建容器

第6集 玩转Docker容器常见命令实战

简介：掌握Docker容器常见命令

- 常用命令（安装部署好Docker后，执行的命令是docker开头），xxx是镜像名称
- 搜索镜像： docker search xxx
- 列出当前系统存在的镜像： docker images
- 拉取镜像： docker pull xxx
 - xxx是具体某个镜像名称(格式 REPOSITORY:TAG)
 - REPOSITORY： 表示镜像的仓库源,TAG： 镜像的标签
- 运行一个容器：

```
docker run --name nginx-xd -p 8080:80 -d nginx
```

```
docker run - 运行一个容器  
          -d 后台运行  
          -p 端口映射  
          --name "xxx" 容器名称
```

- 列举当前运行的容器: docker ps
- 检查容器内部信息: docker inspect 容器名称
- 删除镜像: docker rmi IMAGE_NAME
 - 强制移除镜像不管是否有容器使用该镜像 增加 -f 参数
- 停止某个容器: docker stop 容器名称
- 启动某个容器: docker start 容器名称
- 移除某个容器: docker rm 容器名称 (容器必须是停止状态)
- 列举全部 容器 : docker ps -a
- 查看容器启动日志
 - docker logs -f containerid

第7集 不同系统Docker安装常见问题讲解和解决思路

简介：常见系统安装Docker和一些坑

- 如果没使用阿里云，本地需要安装Docker,才可以进行打包，但是容易出现兼容性问题，大家自行解决
 - Win7~Win10
 - Mac
 - Linux（系统镜像不可能每个人都统一的，所以大家结合百度博文看看）
 - CentOS
 - ubuntu
 - 官方地址
 - <https://docs.docker.com/docker-for-mac/install/>
 - <https://docs.docker.com/docker-for-windows/install/>
- 问题
 - 直接安装Docker不成功，或者下载Yum失败，这个只能根据报错百度检索信息
 - 镜像下载慢

- 搜索修改镜像仓库地址：阿里云、网易云等都有镜像仓库地址（不熟悉不建议乱修改）
 - 本地网络差，下载包容易超时或者慢（只能等）
- 常规的部署只是Docker的冰山一角，课程快速入门，如果想深入可以看我们的Docker专题



小滴课堂

愿景："让编程不再难学，让技术与生活更加有趣"

第五章 新版RabbitMQ安装和web管控台讲解

第1集 基于Linux服务器安装RabbitMQ容器化部署

简介：Docker安装RabbitMQ消息队列

- 登录个人的Linux服务器
 - ssh root@10.211.55.13
- Docker安装RabbitMQ
 - 地址：https://hub.docker.com/_/rabbitmq/

```
#拉取镜像
docker pull rabbitmq:management

docker run -d --hostname rabbit_host1 --name
xd_rabbit -e RABBITMQ_DEFAULT_USER=admin -e
RABBITMQ_DEFAULT_PASS=password -p 15672:15672
-p 5672:5672 rabbitmq:management
```

```
#介绍
-d 以守护进程方式在后台运行
-p 15672:15672 management 界面管理访问端口
-p 5672:5672 amqp 访问端口
--name: 指定容器名
```

--hostname: 设定容器的主机名，它会被写到容器内的 /etc/hostname 和 /etc/hosts，作为容器主机IP的别名，并且将显示在容器的bash中

-e 参数

RABBITMQ_DEFAULT_USER 用户名

RABBITMQ_DEFAULT_PASS 密码

- 主要端口介绍

4369 erlang 发现口

5672 client 端通信口

15672 管理界面 ui 端口

25672 server 间内部通信口

- 访问管理界面

- ip:15672

- 注意事项！！！！

- Linux服务器检查防火墙是否关闭
 - 云服务器检查网络安全组是否开放端口

CentOS 7 以上默认使用的是firewall作为防火墙
查看防火墙状态

```
firewall-cmd --state
```

停止firewall

```
systemctl stop firewalld.service
```

禁止firewall开机启动

```
systemctl disable firewalld.service
```

第2集 深入浅出RabbitMQ的Web管控台介绍

简介： RabbitMQ控制台介绍

● 管控台介绍

- 默认rabbitmq账号密码 guest/guest
- admin/password

Refreshed 2023-01-08 11:12:56 Refresh every 5 seconds

Virtual host: All Cluster rabbit@rabbit_host1 User admin Log out

mq 节点名称

账号信息

相关连接数据

节点内存 磁盘

Name	File descriptors	Socket descriptors	Erlang processes	Memory	Disk space	Uptime	Info	Reset stats
rabbit@rabbit_host1	33	0	553	103 MiB	44 GiB	3h 0m	basic disc rss	This node All nodes
1048576 available	943629 available	1048576 available	733 MiB high watermark MiB low watermark					

Churn statistics

相关速率信息, 检测和监控

Connection operations last minute

Created: 0.00/s
Closed: 0.00/s

Channel operations last minute

Created: 0.00/s
Closed: 0.00/s

Queue operations last minute

Declared: 0.00/s
Created: 0.00/s

应用 Google 百度 翻译 学习 person 架构技术 xdclass.net 小课堂 电商资讯

Refreshed 2023-01-08 11:15:56 Refresh every 5 seconds

Virtual host: All Cluster rabbit@rabbit_host1 User admin Log out

mq 节点名称

账号信息

不同场景的端口

amqp 操作端口

集群通信端口

管控台端口

Ports and contexts

不同场景的端口

amqp 操作端口

集群通信端口

管控台端口

Web contexts

Context	Bound to	Port	SSL	Path
RabbitMQ Management	0.0.0.0	15672	○	/
RabbitMQ Prometheus	0.0.0.0	15692	○	/

Export definitions

Filename for download: rabbit_rabbit_host1_2 迁移数据导出信息

Import definitions

Definitions file: 迁移数据导入信息

选择文件 未选择任何文件

Upload broker definitions

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

● 每个虚拟主机默认就有7个交换机

Exchanges

▼ All exchanges (7)

Pagination

Page 1 of 1 - Filter: Regex ?

Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
/	(AMQP default)	direct	D	0.00/s	0.00/s	
/	amq.direct	direct	D			
/	amq.fanout	fanout	D			
/	amq.headers	headers	D			
/	amq.match	headers	D			
/	amq.rabbitmq.trace	topic	D I			
/	amq.topic	topic	D			

▶ Add a new exchange



小滴课堂

趣"

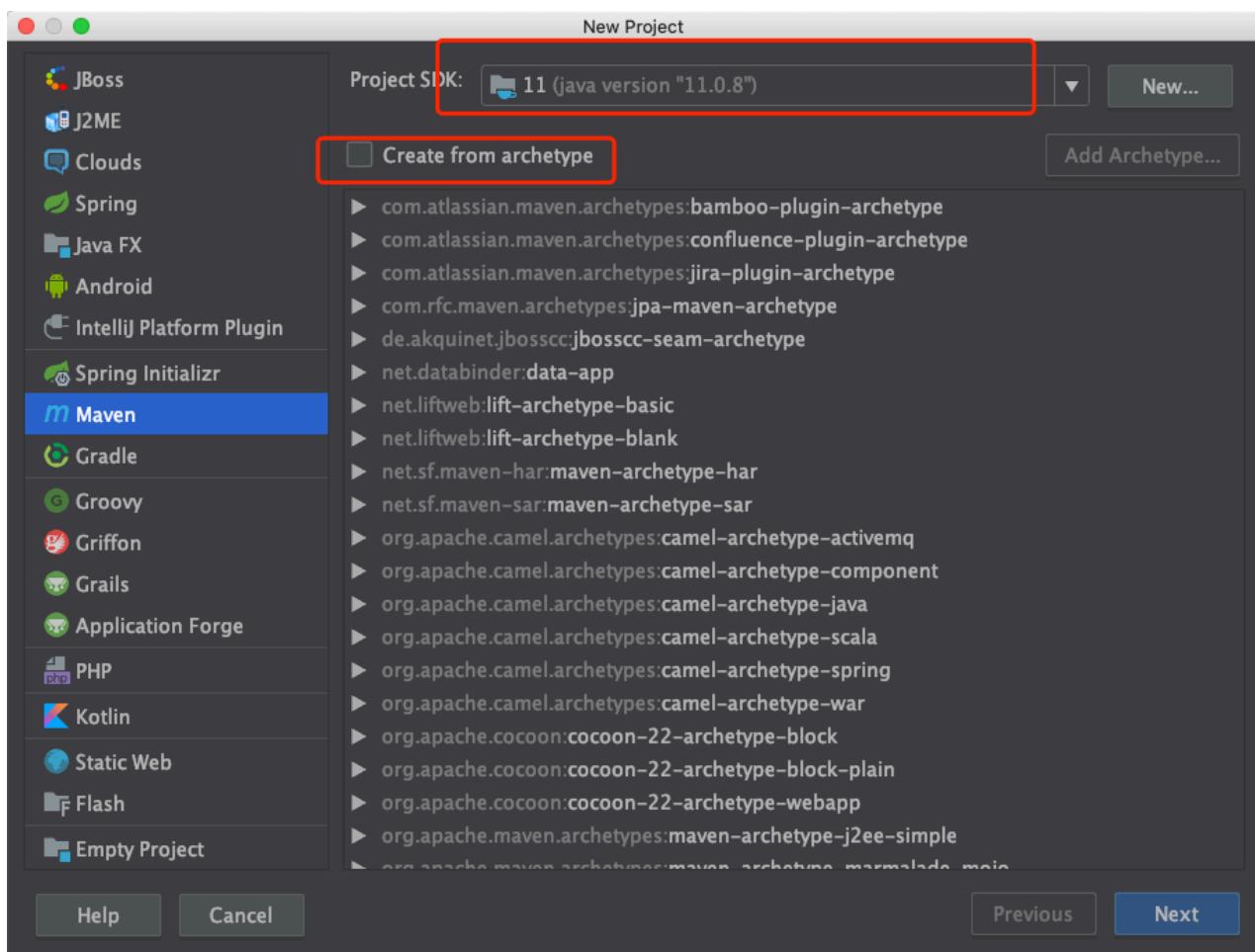
愿景："让编程不再难学，让技术与生活更加有

第六章 玩转Java整合RabbitMQ 工作队列模型实战

第1集 Java项目创建整合RabbitMQ

简介：Java项目创建并整合RabbitMQ

- 创建Maven项目



- 项目创建需要点时间-大家静待就行
 - 删除部分没用的配置
 - 修改jdk版本，如果是使用jdk8的，改为1.8即可

- 添加依赖
 - 官方地址: <https://www.rabbitmq.com/java-client.html>
 - 依赖地址: <https://mvnrepository.com/artifact/com.rabbitmq/amqp-client/5.10.0>

```
<properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>

<maven.compiler.source>11</maven.compiler.sourc
e>

<maven.compiler.target>11</maven.compiler.targe
t>

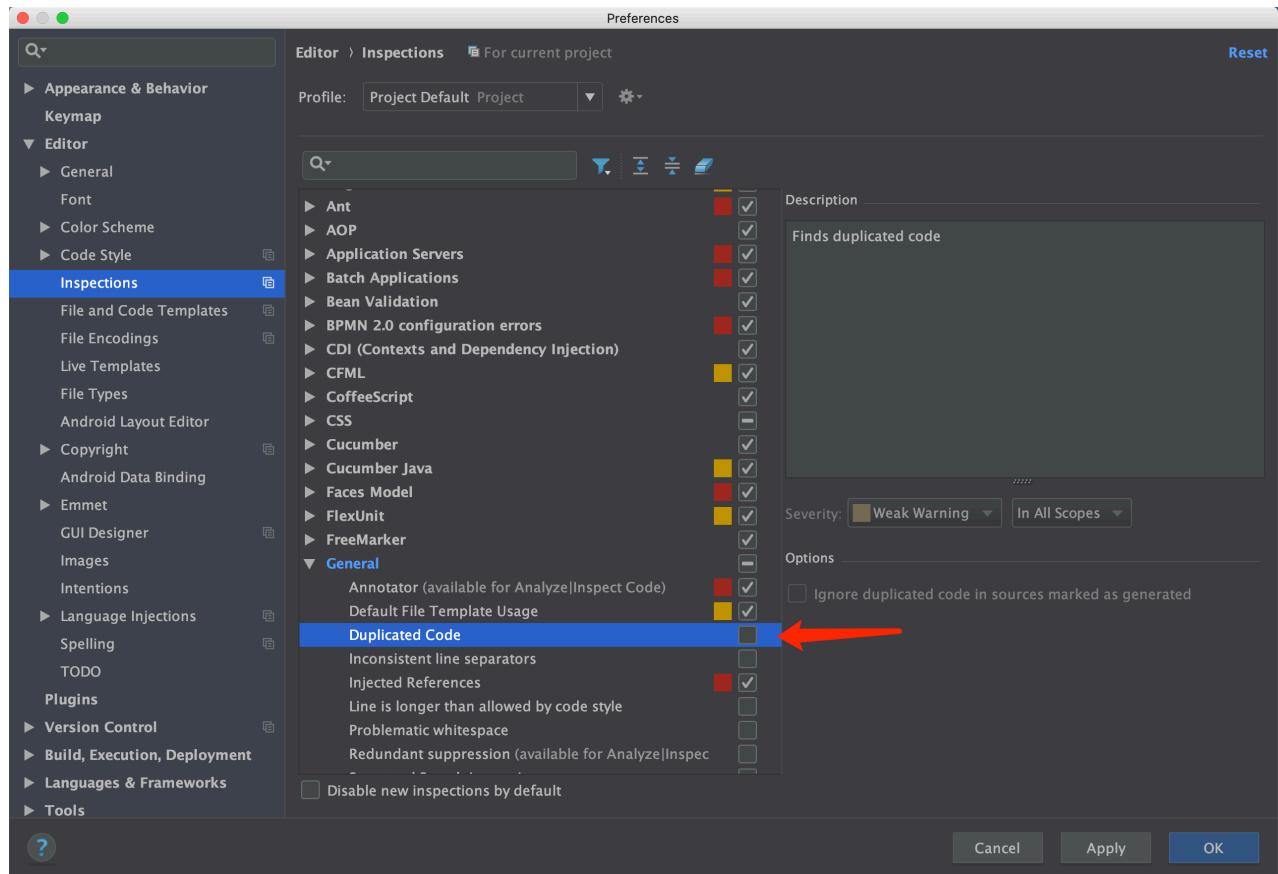
</properties>

<dependencies>

    <dependency>
        <groupId>com.rabbitmq</groupId>
        <artifactId>amqp-client</artifactId>
        <version>5.10.0</version>
    </dependency>

</dependencies>
```

- 涉及重复代码，关闭idea重复代码检测



第2集 玩转RabbitMQ简单队列实战

简介：玩转RabbitMQ简单队列实战

- 简单队列测试
 - 文档
 - <https://www.rabbitmq.com/tutorials/tutorial-one-java.html>



- 消息生产者

```
public class Send {  
  
    private final static String QUEUE_NAME =  
    "hello";  
  
    public static void main(String[] argv)  
    throws Exception {  
        ConnectionFactory factory = new  
        ConnectionFactory();  
        factory.setHost("10.211.55.13");  
    }  
}
```

```
factory.setUsername("admin");
factory.setPassword("password");
factory.setVirtualHost("/dev");
factory.setPort(5672);
try ( //JDK7语法 或自动关闭 connection
和channel
        //创建连接
        Connection connection =
factory.newConnection();
        //创建信道
        Channel channel =
connection.createChannel()) {
    /**
     * 队列名称
     * 持久化配置：mq重启后还在
     * 是否独占：只能有一个消费者监听队列；当
connection关闭是否删除队列，一般是false，发布订阅是独
占
     * 自动删除：当没有消费者的时候，自动删除
掉，一般是false
     * 其他参数
     *
     * 队列不存在则会自动创建，如果存在则不会
覆盖，所以此时的时候需要注意属性
    */
    channel.queueDeclare(QUEUE_NAME,
false, false, false, null);
    String message = "Hello World!";
    /**

```

```

        * 参数说明：
        * 交换机名称：不写则是默认的交换机，那路由健需要和队列名称一样才可以被路由，
        * 路由健名称
        * 配置信息
        * 发送的消息数据：字节数组
        */
    channel.basicPublish("", QUEUE_NAME, null,
message.getBytes(StandardCharsets.UTF_8));
    System.out.println(" [x] Sent '" +
message + "'");
}
}
}

```

- 消息消费者（会一直监听队列）

```

public class Recv {

    private final static String QUEUE_NAME =
"hello";

    public static void main(String[] argv)
throws Exception {
    ConnectionFactory factory = new
ConnectionFactory();
    factory.setHost("10.211.55.13");
    factory.setUsername("admin");
}
}

```

```
factory.setPassword("password");
factory.setVirtualHost("/xdclass1");
factory.setPort(5672);

//消费者一般不增加自动关闭
Connection connection =
factory.newConnection();
Channel channel =
connection.createChannel();

channel.queueDeclare(QUEUE_NAME, false,
false, false, null);
System.out.println(" [*] Waiting for
messages. To exit press CTRL+C");

//回调方法，下面两种都行
Consumer consumer = new
DefaultConsumer(channel){
    @Override
    public void handleDelivery(String
consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[ ] body)
throws IOException {
        // consumerTag 是固定的 可以做此会
话的名字， deliveryTag 每次接收消息+1
        System.out.println("consumerTag
消息标识="+consumerTag);
        //可以获取交换机，路由键等
    }
}
```

```
        System.out.println("envelope元数
据="+envelope);

        System.out.println("properties
配置信息="+properties);

        System.out.println("body="+new
String(body,"utf-8")));
    }

};

channel.basicConsume(QUEUE_NAME,true,consumer);

//          DeliverCallback deliverCallback =
(consumerTag, envelop, delivery,properties,
msg) -> {
//          String message = new String(msg,
"UTF-8");
//          System.out.println(" [x] Received
'" + message + "' ");
//      };

//自动确认消息
//channel.basicConsume(QUEUE_NAME,
true, deliverCallback, consumerTag -> { });
}

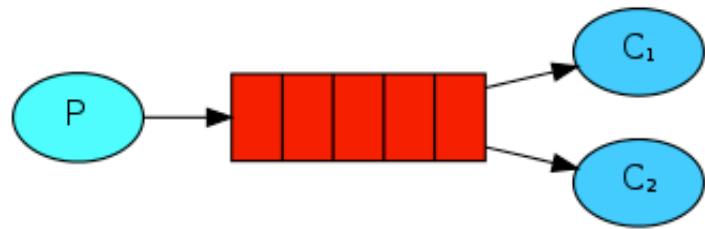
}
```

第3集 玩转RabbitMQ工作队列 轮训策略实战

简介： RabbitMQ工作队列 轮训策略讲解实战

- 工作队列
 - 消息生产能力大于消费能力， 增加多几个消费节点
 - 和简单队列类似， 增加多个几个消费节点， 处于竞争关系

- 默认策略：round robin 轮训



```
public class Send {  
  
    private final static String QUEUE_NAME =  
    "work_mq_rr";  
  
    public static void main(String[] argv)  
    throws Exception {  
        ConnectionFactory factory = new  
        ConnectionFactory();  
        factory.setHost("10.211.55.13");  
        factory.setUsername("admin");  
        factory.setPassword("password");  
        factory.setVirtualHost("/dev");  
        factory.setPort(5672);  
  
        try {  
            //创建连接  
            Connection connection =  
            factory.newConnection();  
            //创建信道  
            Channel channel =  
            connection.createChannel() {
```

```
    /**
     * 队列名称
     * 持久化配置
     * 排他配置
     * 自动删除
     * 其他参数
    */
    channel.queueDeclare(QUEUE_NAME,
false, false, false, null);
    //轮训发送 10个
    for(int i=0;i<10;i++){
        String message = "Hello
World!" + i;
        channel.basicPublish("", QUEUE_NAME, null,
message.getBytes(StandardCharsets.UTF_8));
        System.out.println(" [x] Sent
" + message + " ");
    }
}
```

● 消费者代码1

```
public class Recv1 {

    private final static String QUEUE_NAME =
"work_mq_rr";
```

```
public static void main(String[] argv)
throws Exception {
    ConnectionFactory factory = new
ConnectionFactory();
    factory.setHost("10.211.55.13");
    factory.setUsername("admin");
    factory.setPassword("password");
    factory.setVirtualHost("/xdclass1");
    factory.setPort(5672);

    Connection connection =
factory.newConnection();
    Channel channel =
connection.createChannel();

    channel.queueDeclare(QUEUE_NAME, false,
false, false, null);
    System.out.println(" [*]Waiting for
messages. To exit press CTRL+C");

    DeliverCallback deliverCallback =
(consumerTag, delivery) -> {
        //模拟消费缓慢
        try {
            TimeUnit.SECONDS.sleep(3);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
        }

        String message = new
String(delivery.getBody(), "UTF-8");
        System.out.println("[x] Received '" +
message + "'");

        //手工确认消息消费，不是多条确认

channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
};

//关闭自动确认消息
channel.basicConsume(QUEUE_NAME, false,
deliverCallback, consumerTag -> { });

}
```

- 消费者代码2

```
public class Recv2 {

    private final static String QUEUE_NAME =
"work_mq_rr";

    public static void main(String[] argv)
throws Exception {
```

```
ConnectionFactory factory = new
ConnectionFactory();
    factory.setHost("10.211.55.13");
    factory.setUsername("admin");
    factory.setPassword("password");
    factory.setVirtualHost("/xdclass1");
    factory.setPort(5672);

    Connection connection =
factory.newConnection();
    Channel channel =
connection.createChannel();

    channel.queueDeclare(QUEUE_NAME, false,
false, false, null);
    System.out.println(" [*] Waiting for
messages. To exit press CTRL+C");

    DeliverCallback deliverCallback =
(consumerTag, delivery) -> {
        //模拟消费缓慢
        try {
            TimeUnit.SECONDS.sleep(3);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        String message = new
String(delivery.getBody(), "UTF-8");
```

```
        System.out.println(" [x] Received  
        " + message + "'");  
  
        //手工确认消息消费，不是多条确认  
  
    channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
};  
  
//关闭自动确认消息  
channel.basicConsume(QUEUE_NAME, false,  
deliverCallback, consumerTag -> { });  
}  
}
```

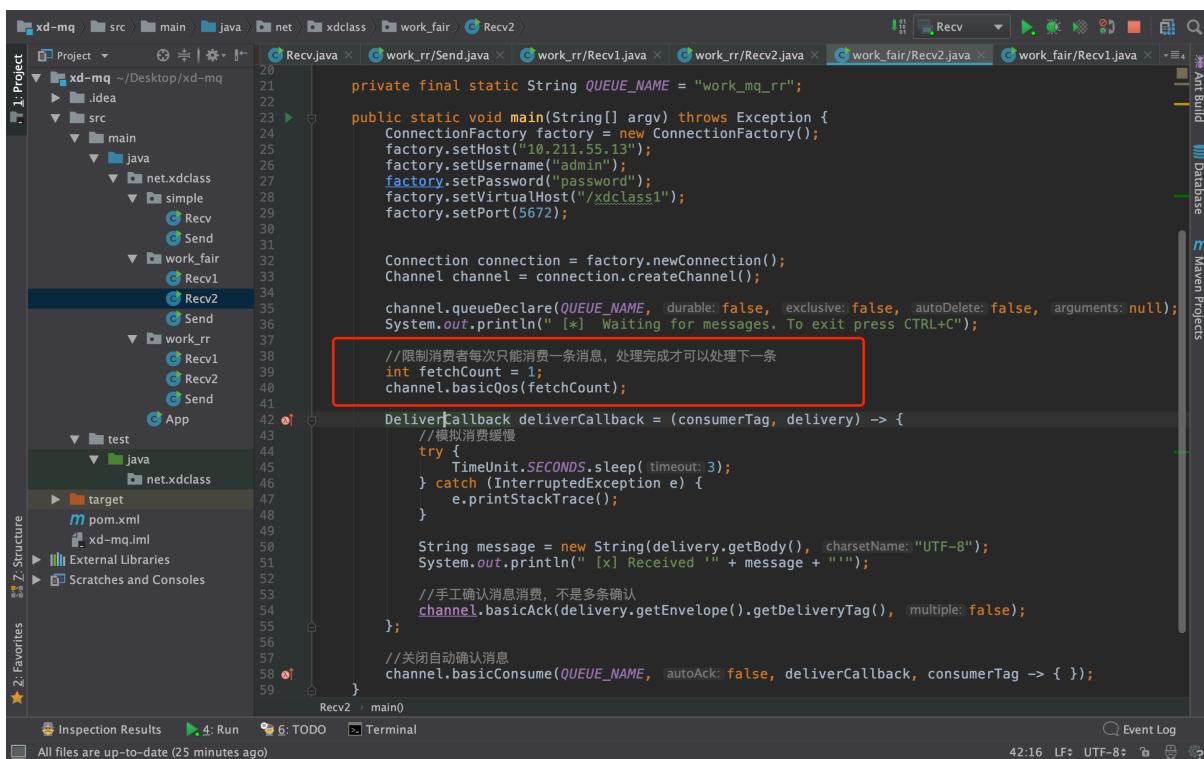
- 轮训策略验证

- 先启动两个消费者，再启动生产者
- 缺点：存在部分节点消费过快，部分节点消费慢，导致不能合理处理消息

第4集 玩转RabbitMQ工作队列 公平策略实战

简介： RabbitMQ工作队列 公平策略讲解实战

- 公平策略验证
 - 修改消费者策略
 - 解决消费者能力消费不足的问题，降低消费时间问题



```
private final static String QUEUE_NAME = "work_mq_rr";  
public static void main(String[] argv) throws Exception {  
    ConnectionFactory factory = new ConnectionFactory();  
    factory.setHost("10.211.55.13");  
    factory.setUsername("admin");  
    factory.setPassword("password");  
    factory.setVirtualHost("//xdclass1");  
    factory.setPort(5672);  
  
    Connection connection = factory.newConnection();  
    Channel channel = connection.createChannel();  
  
    channel.queueDeclare(QUEUE_NAME, durable: false, exclusive: false, autoDelete: false, arguments: null);  
    System.out.println(" [*] Waiting for messages. To exit press CTRL+C");  
  
    //限制消费者每次只能消费一条消息，处理完成才可以处理下一条  
    int fetchCount = 1;  
    channel.basicQos(fetchCount);  
  
    DeliverCallback deliverCallback = (consumerTag, delivery) -> {  
        //模拟消费缓慢  
        try {  
            TimeUnit.SECONDS.sleep( timeout: 3);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
  
        String message = new String(delivery.getBody(), charsetName: "UTF-8");  
        System.out.println(" [x] Received '" + message + "'");  
  
        //手工确认消息消费，不是多条确认  
        channel.basicAck(delivery.getEnvelope().getDeliveryTag(), multiple: false);  
    };  
  
    //关闭自动确认消息  
    channel.basicConsume(QUEUE_NAME, autoAck: false, deliverCallback, consumerTag -> { });  
}  
Recv2 : main()
```



小滴课堂 愿景："让编程不再难学，让技术与生活更加有趣"

第七章 玩转RabbitMQ 交换机和发布订阅模型实战

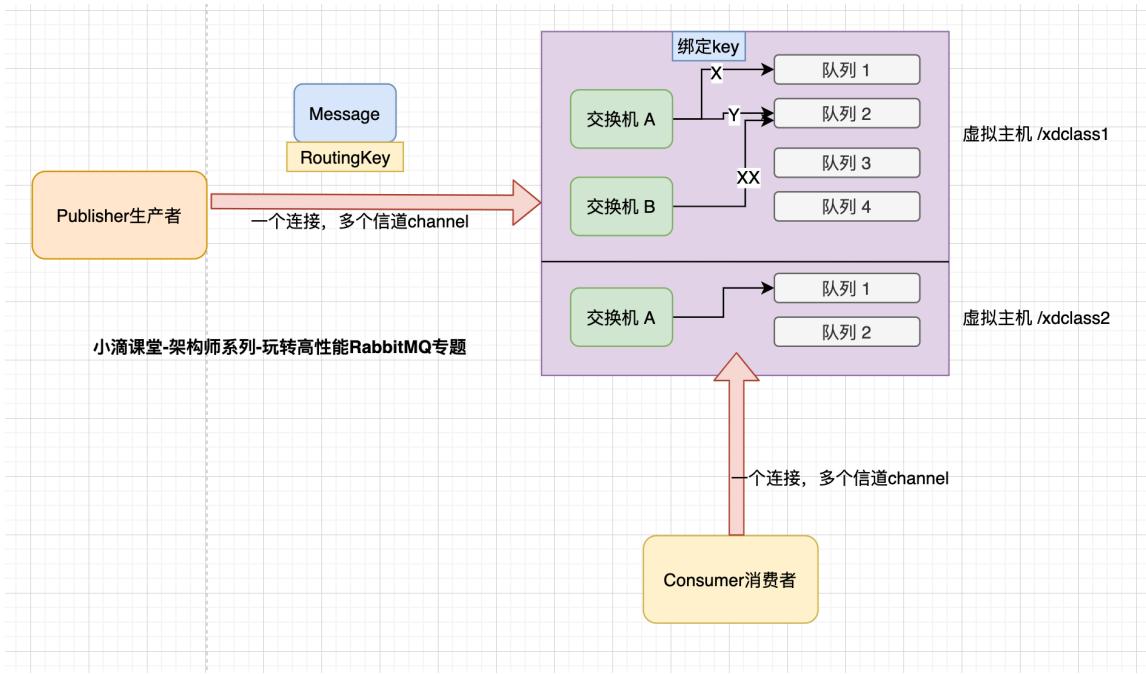
第1集 RabbitMQ的常见的Exchange交换机介绍

简介： RabbitMQ的Exchange交换机介绍

- RabbitMQ的Exchange 交换机



- 生产者将消息发送到 Exchange，交换器将消息路由到一个或者多个队列中，交换机有多个类型，队列和交换机是多对多的关系。
- 交换机只负责转发消息，不具备存储消息的能力，如果没有队列和exchange绑定，或者没有符合的路由规则，则消息会被丢失
- RabbitMQ有四种交换机类型，分别是Direct exchange、Fanout exchange、Topic exchange、Headers exchange，最后的基本不用



- 交换机类型

- Direct Exchange 定向
 - 将一个队列绑定到交换机上，要求该消息与一个特定的路由键完全匹配
 - 例子：如果一个队列绑定到该交换机上要求路由键“aabb”，则只有被标记为“aabb”的消息才被转发，不会转发aabb.cc，也不会转发gg.aabb，只会转发aabb
 - 处理路由健
- Fanout Exchange 广播
 - 只需要简单的将队列绑定到交换机上，一个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。很像子网广播，每台子网内的主机都获得了一份复制的消息
 - Fanout交换机转发消息是最快的，用于发布订阅，广播形式，中文是扇形

- 不处理路由键
- Topic Exchange 通配符
 - 主题交换机是一种发布/订阅的模式，结合了直连交换机与扇形交换机的特点
 - 将路由键和某模式进行匹配。此时队列需要绑定要一个模式上
 - 符号“#”匹配一个或多个词，符号“*”匹配不多不少一个词
 - 例子：因此“abc.#”能够匹配到“abc.def.ghi”，但是“abc.*”只会匹配到“abc.def”。
- Headers Exchanges (少用)
 - 根据发送的消息内容中的headers属性进行匹配，在绑定Queue与Exchange时指定一组键值对
 - 当消息发送到RabbitMQ时会取到该消息的headers与Exchange绑定时指定的键值对进行匹配；
 - 如果完全匹配则消息会路由到该队列，否则不会路由到该队列
 - 不处理路由键

Exchanges

▼ All exchanges (8)

Pagination

Page of 1 - Filter: Regex ?

其他7个是默认自带的

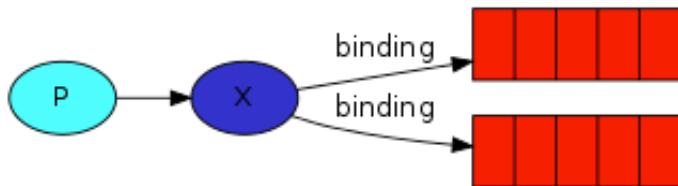
Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
/xdclass1	(AMQP default)	direct	D	0.00/s	0.00/s	
/xdclass1	amq.direct	direct	D			
/xdclass1	amq.fanout	fanout	D			
/xdclass1	amq.headers	headers	D			
/xdclass1	amq.match	headers	D			
/xdclass1	amq.rabbitmq.trace	topic	D I			
/xdclass1	amq.topic	topic	D			
/xdclass1	exchange_fanout	fanout		0.00/s	0.00/s	

▶ Add a new exchange

第2集 RabbitMQ的发布订阅消息模型介绍

简介：RabbitMQ的发布订阅消息模型介绍

- 什么是rabbitmq的发布订阅模式
 - 发布-订阅模型中，消息生产者不再是直接面对queue(队列名称)，而是直面exchange,都需要经过exchange来进行消息的发送,所有发往同一个fanout交换机的消息都会被所有监听这个交换机的消费者接收到
 - 发布订阅-消息模型引入fanout交换机
 - 文档：<https://www.rabbitmq.com/tutorials/tutorial-three-java.html>
- 发布订阅模型应用场景
 - 微信公众号
 - 新浪微博关注



- rabbitmq发布订阅模型
 - 通过把消息发送给交换机，交互机转发给对应绑定的队列
 - 交换机绑定的队列是排它独占队列，自动删除

fanout 广播交换机绑定, 排它队列

Virtual host	Name	Type	Features	State	Messages			Message rates			+/-
					Ready	Unacked	Total	incoming	deliver / get	ack	
/xdclass1	amq.gen-IkSH3K7RbjJsmWUpRHBeqq	classic	AD Excl	idle	0	0	0	0.00/s	0.00/s	0.00/s	
/xdclass1	amq.gen-oS0xK7KSkyUczTSUj7IQ5g	classic	AD Ex	exclusive: true	0	0	0	0.00/s	0.00/s	0.00/s	
/xdclass1	hello	classic		idle	0	0	0	0.00/s	0.00/s	0.00/s	
/xdclass1	work_mq_rr	classic		idle	0	0	0	0.00/s	0.00/s	0.00/s	

▶ Add a new queue

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

第3集 RabbitMQ的发布订阅消息模型代码实战

简介： RabbitMQ的发布订阅消息模型代码实战

- 发送端

```
public class Send {  
  
    private final static String EXCHANGE_NAME =  
"exchange_fanout";  
  
    public static void main(String[ ] argv)  
throws Exception {  
    ConnectionFactory factory = new  
ConnectionFactory();  
    factory.setHost("10.211.55.13");  
    factory.setUsername("admin");  
    factory.setPassword("password");  
    factory.setVirtualHost("/dev");  
    factory.setPort(5672);  
  
    /**  
     * 消息生产者不用过多操作，只需要和交换机绑定  
     * 即可  
     */  
    try ( //创建连接  
        Connection connection =  
factory.newConnection();  
        //创建信道  
        Channel channel =  
connection.createChannel()) {  
  
        //绑定交换机, fanout扇形，即广播类型
```

```

channel.exchangeDeclare(EXCHANGE_NAME,BuiltinExchangeType.FANOUT);

        String message = "Hello World pub
!";
        channel.basicPublish(EXCHANGE_NAME,
        "", null,
message.getBytes(StandardCharsets.UTF_8));
        System.out.println(" [x] Sent '" +
message + "'");

    }

}

```

- 消费端（两个节点）

```

public class Recv1 {

    private final static String EXCHANGE_NAME =
"exchange_fanout";

    public static void main(String[ ] argv)
throws Exception {
    ConnectionFactory factory = new
ConnectionFactory();
    factory.setHost("10.211.55.13");
    factory.setUsername("admin");

```

```
factory.setPassword("password");
factory.setVirtualHost("/dev");
factory.setPort(5672);

//消费者一般不增加自动关闭
Connection connection =
factory.newConnection();
Channel channel =
connection.createChannel();

//绑定交换机,fanout扇形, 即广播类型

channel.exchangeDeclare(EXCHANGE_NAME,BuiltinExchangeType.FANOUT);

//获取队列 (排它队列)
String queueName =
channel.queueDeclare().getQueue();

//绑定队列和交换机,fanout交换机不用指定
routingkey

channel.queueBind(queueName,EXCHANGE_NAME,"");

DeliverCallback deliverCallback =
(consumerTag, delivery) -> {
    String message = new
String(delivery.getBody(), "UTF-8");
```

```
        System.out.println(" [x] Received  
        " + message + "'");  
    };  
  
    //自动确认消息  
    channel.basicConsume(queueName, true,  
deliverCallback, consumerTag -> { });  
  
}  
}
```

- 验证：启动两个消费者，一个生产者发送消息



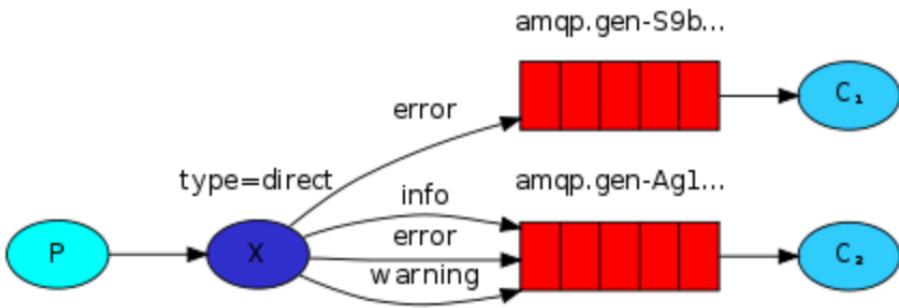
愿景："让编程不再难学，让技术与生活更加有趣"

第八章 玩转RabbitMQ 路由、主题模式 实战和总结

第1集 玩转RabbitMQ的路由模式和应用场景

简介： RabbitMQ的路由模式和应用场景

- 什么是rabbitmq的路由模式
 - 文档：<https://www.rabbitmq.com/tutorials/tutorial-four-java.html>
 - 交换机类型是Direct
 - 队列和交换机绑定，需要指定一个路由key(也叫Bingding Key)
 - 消息生产者发送消息给交换机，需要指定routingKey
 - 交换机根据消息的路由key，转发给对应的队列



- 例子：日志采集系统 ELK
 - 一个队列收集error信息-》 告警
 - 一个队列收集全部信息-》 日常使用

第2集 RabbitMQ的路由模式代码实战

简介：RabbitMQ路由模式代码实战

- 消息生产者

```
public class Send {  
  
    private final static String EXCHANGE_NAME =  
"exchange_direct";  
  
    public static void main(String[] argv)  
throws Exception {  
    ConnectionFactory factory = new  
ConnectionFactory();  
    factory.setHost("10.211.55.13");  
    factory.setUsername("admin");  
    factory.setPassword("password");  
    factory.setVirtualHost("/xdclass1");  
    factory.setPort(5672);  
  
    /**  
     * 消息生产者不用过多操作，只需要和交换机绑定  
     * 即可  
     */  
  
    try ( //创建连接
```

```
        Connection connection =
factory.newConnection();
        //创建信道
        Channel channel =
connection.createChannel() {
        //绑定交换机，直连交换机
channel.exchangeDeclare(EXCHANGE_NAME,BuiltinExchangeType.DIRECT);

        String error = "我是错误日志";
        String info = "我是info日志";
        String debug = "我是debug日志";
        channel.basicPublish(EXCHANGE_NAME,
"errorRoutingKey", null,
error.getBytes(StandardCharsets.UTF_8));
        channel.basicPublish(EXCHANGE_NAME,
"infoRoutingKey", null,
info.getBytes(StandardCharsets.UTF_8));
        channel.basicPublish(EXCHANGE_NAME,
"debugRoutingKey", null,
debug.getBytes(StandardCharsets.UTF_8));

        System.out.println("消息发送成功");
    }
}
```

- 消息消费者（两个节点）

```
public class Recv1 {  
  
    private final static String EXCHANGE_NAME =  
"exchange_direct";  
  
    public static void main(String[] argv)  
throws Exception {  
    ConnectionFactory factory = new  
ConnectionFactory();  
    factory.setHost("10.211.55.13");  
    factory.setUsername("admin");  
    factory.setPassword("password");  
    factory.setVirtualHost("/xdclass1");  
    factory.setPort(5672);  
  
    //消费者一般不增加自动关闭  
    Connection connection =  
factory.newConnection();  
    Channel channel =  
connection.createChannel();  
  
    //绑定交换机,fanout扇形, 即广播类型  
  
channel.exchangeDeclare(EXCHANGE_NAME,BuiltinEx  
changeType.DIRECT);
```

```
//获取队列
String queueName =
channel.queueDeclare().getQueue();

//绑定队列和交换机，另外一个节点只绑定一个
errorRoutingKey

channel.queueBind(queueName, EXCHANGE_NAME, "erro
rRoutingKey");

channel.queueBind(queueName, EXCHANGE_NAME, "info
RoutingKey");

channel.queueBind(queueName, EXCHANGE_NAME, "debu
gRoutingKey");

DeliverCallback deliverCallback =
(consumerTag, delivery) -> {
    String message = new
String(delivery.getBody(), "UTF-8");
    System.out.println(" [x] Received
'" + message + "' ");
};

//自动确认消息
channel.basicConsume(queueName, true,
deliverCallback, consumerTag -> { });
```

```
}
```

```
}
```

RabbitMQ™ RabbitMQ 3.8.9 Erlang 23.2.1

Overview Connections Channels Exchanges Queues Admin

Exchange: exchange_direct in virtual host /xdclass1

▼ Overview

Message rates last minute ?



Details

Type direct

Features

Policy

▼ Bindings

This exchange

前3个路由key绑定到同一个队列,
第4个路由key进到另外1个队列

To	Routing key	Arguments	
amq.gen-WKPE_XLn4lzMotGp88chng	debugRoutingKey		<button>Unbind</button>
amq.gen-WKPE_XLn4lzMotGp88chng	errorRoutingKey		<button>Unbind</button>
amq.gen-WKPE_XLn4lzMotGp88chng	infoRoutingKey		<button>Unbind</button>
amq.gen-bG1PTMoxC_UmRh0QS_jGUg	errorRoutingKey		<button>Unbind</button>

Add binding from this exchange

To queue : *

Routing key:

Exchanges

▼ All exchanges (9)

Pagination

Page of 1 - Filter: Regex ?

Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
/xdclass1	(AMQP default)	direct	D	0.00/s	0.00/s	
/xdclass1	amq.direct	direct	D			
/xdclass1	amq.fanout	fanout	D			
/xdclass1	amq.headers	headers	D			
/xdclass1	amq.match	headers	D			
/xdclass1	amq.rabbitmq.trace	topic	D I			
/xdclass1	amq.topic	topic	D			
/xdclass1	exchange_direct	direct		0.00/s	0.00/s	
/xdclass1	exchange_fanout	fanout		0.00/s	0.00/s	

▶ Add a new exchange

第3集 玩转RabbitMQ的topic主题通配符模式和应用场景

简介： RabbitMQ的主题模式和应用场景

- 背景：
 - 如果业务很多路由key，怎么维护？？
 - topic交换机，支持通配符匹配模式，更加强大
 - 工作基本都是用这个topic模式
- 什么是rabbitmq的主题模式
 - 文档 <https://www.rabbitmq.com/tutorials/tutorial-finding-queues.html>
 - 交换机是 topic, 可以实现发布订阅模式fanout和路由模式Direct 的功能，更加灵活，支持模式匹配，通配符等
 - 交换机同过通配符进行转发到对应的队列，* 代表一个词，#代表1个或多个词，一般用#作为通配符居多，比如#.order, 会匹配 info.order、sys.error.order, 而*.order，只会匹配 info.order, 之间是使用. 点进行分割多个词的；如果是.，则info.order、error.order都会匹配
 - 注意
 - 交换机和队列绑定时用的binding使用通配符的路由健
 - 生产者发送消息时需要使用具体的路由健
- 测试,下面的匹配规则是怎样的

quick.orange.rabbit 只会匹配 *.orange.* 和 *.*.rabbit , 进到Q1和Q2

lazy.orange.elephant 只会匹配 *.orange.* 和 lazy.#, 进到Q1和Q2

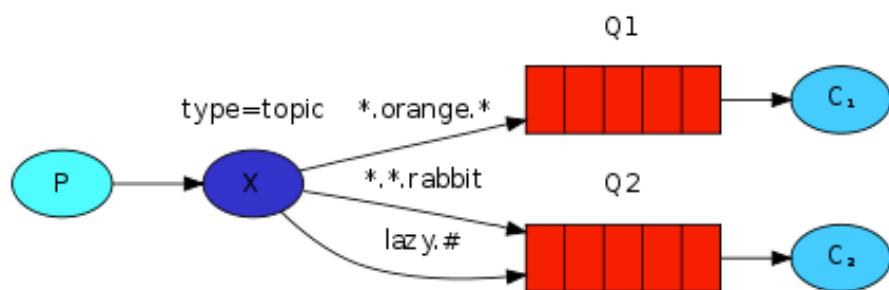
quick.orange.fox 只会匹配 *.orange.* , 进入Q1

lazy.brown.fox 只会匹配lazy.#, 进入Q2

lazy.pink.rabbit 只会匹配 lazy.#和*.*.rabbit , 同个队列进入Q2(消息只会发一次)

quick.brown.fox 没有匹配, 默认会被丢弃, 可以通过回调监听二次处理

lazy.orange.male.rabbit, 只会匹配 lazy.#, 进入Q2



- 例子：日志采集系统

- 一个队列收集订单系统的全部日志信息, order.log.#
- 一个队列收集全部系统的全部日志信息, #.log

第4集 玩转RabbitMQ的topic主题通配符模式 代码实战

简介： RabbitMQ的topic主题模式代码实战

- 例子： 日志采集系统
 - 一个队列收集订单系统的error日志信息，
`order.log.error`
 - 一个队列收集全部系统的全部级别日志信息, `* .log. *`
- 生产者

```
public class Send {

    private final static String EXCHANGE_NAME =
"exchange_topic";

    public static void main(String[ ] argv)
throws Exception {
        ConnectionFactory factory = new
ConnectionFactory();
        factory.setHost("10.211.55.13");
        factory.setUsername("admin");
        factory.setPassword("password");
        factory.setVirtualHost("/dev");
        factory.setPort(5672);

        /**
         * 消息生产者不用过多操作，只需要和交换机绑定
         * 即可
         */
try ( //创建连接
        Connection connection =
factory.newConnection();
        //创建信道
        Channel channel =
connection.createChannel() {
        //绑定交换机，直连交换机
    }
}
```

```
channel.exchangeDeclare(EXCHANGE_NAME,BuiltinExchangeType.TOPIC);

        String error = "我是订单错误日志";
        String info = "我是订单info日志";
        String debug = "我是商品debug日志";
        channel.basicPublish(EXCHANGE_NAME,
"order.log.error", null,
error.getBytes(StandardCharsets.UTF_8));
        channel.basicPublish(EXCHANGE_NAME,
"order.log.info", null,
info.getBytes(StandardCharsets.UTF_8));
        channel.basicPublish(EXCHANGE_NAME,
"product.log.debug", null,
debug.getBytes(StandardCharsets.UTF_8));

        System.out.println("消息发送成功");

    }
}
```

- 消费者（两个）

```
public class Recv1 {
```

```
private final static String EXCHANGE_NAME =
"exchange_topic";

public static void main(String[ ] argv)
throws Exception {
    ConnectionFactory factory = new
ConnectionFactory();
    factory.setHost("10.211.55.13");
    factory.setUsername("admin");
    factory.setPassword("password");
    factory.setVirtualHost("/dev");
    factory.setPort(5672);

    //消费者一般不增加自动关闭
    Connection connection =
factory.newConnection();
    Channel channel =
connection.createChannel();

    //绑定交换机
channel.exchangeDeclare(EXCHANGE_NAME,BuiltinEx
changeType.TOPIC);

    //获取队列
    String queueName =
channel.queueDeclare().getQueue();

    //绑定队列和交换机, 第一个节点
```

```
//channel.queueBind(queueName,EXCHANGE_NAME,"order.log.error");  
  
        //绑定队列和交换机,第二个节点  
  
        //channel.queueBind(queueName,EXCHANGE_NAME,"*.log.*");  
  
        DeliverCallback deliverCallback =  
(consumerTag, delivery) -> {  
            String message = new  
String(delivery.getBody(), "UTF-8");  
            System.out.println(" [x] Received  
'" + message + "'");  
        };  
  
        //自动确认消息  
        channel.basicConsume(queueName, true,  
deliverCallback, consumerTag -> { });  
  
    }  
}
```

Exchange: exchange_topic in virtual host /xdclass1

▼ Overview

Message rates last minute ?



Details

Type	topic
Features	
Policy	

▼ Bindings

This exchange

两个队列和交换机的bingding key

To	Routing key	Arguments	
amq.gen-20Sbb9OE06DLZr-0Jf52xw	order.#		Unbind
amq.gen-XsOmlkb0r8BU46mBcT-vFg	*.*.debug		Unbind

Add binding from this exchange

To queue	:	<input type="text"/>	*	
Routing key:	<input type="text"/>			
Arguments:	<input type="text"/>	=	<input type="text"/>	String
Bind				

第5集 RabbitMQ的多种工作模式总结

简介：RabbitMQ的多个工作模式总结

- 对照官网回顾总结
 - <https://www.rabbitmq.com/getstarted.html>
- 简单模式
 - 一个生产、一个消费，不用指定交换机，使用默认交换机
- 工作队列模式
 - 一个生产，多个消费，可以有轮训和公平策略，不用指定交换机，使用默认交换机
- 发布订阅模式
 - fanout类型交换机，通过交换机和队列绑定，不用指定绑定的路由键，生产者发送消息到交换机，fanout交换机直接进行转发，消息不用指定routingkey路由键

- 路由模式
 - direct类型交换机，通过交换机和队列绑定，指定绑定的路由键，生产者发送消息到交换机，交换机根据消息的路由key进行转发到对应的队列，消息要指定routingkey路由键
- 通配符模式
 - topic交换机，通过交换机和队列绑定，指定绑定的【通配符路由键】，生产者发送消息到交换机，交换机根据消息的路由key进行转发到对应的队列，消息要指定routingkey路由键



愿景："让编程不再难学，让技术与生活更加有趣"

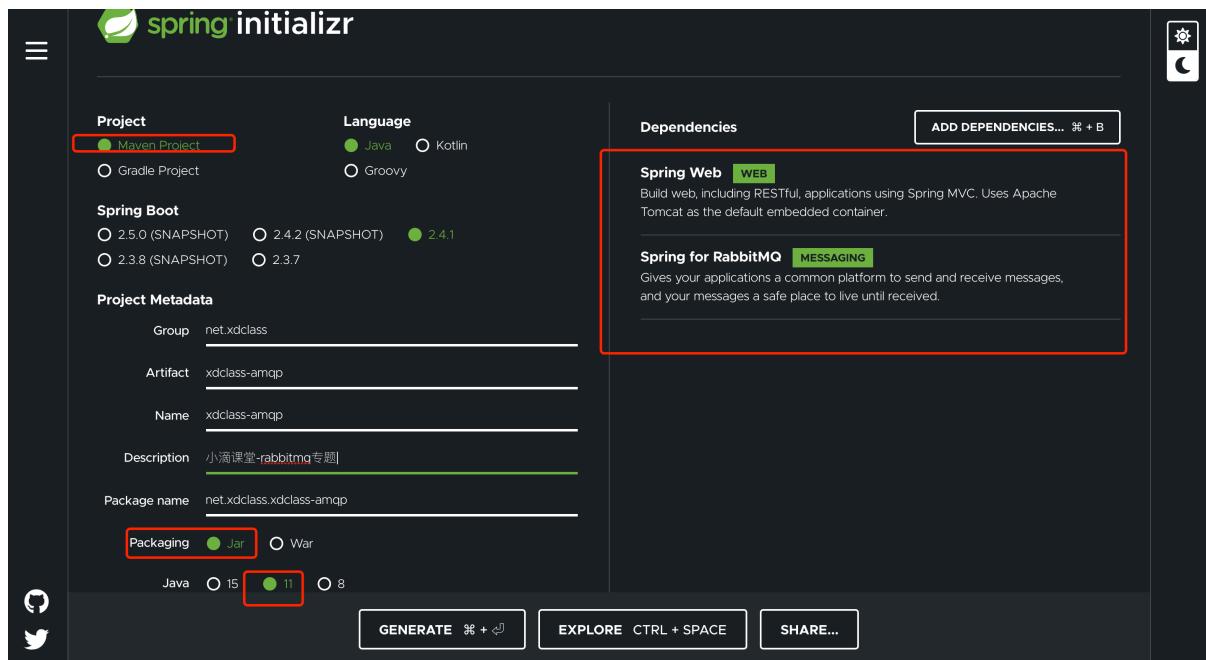
第九章 SpringBoot2.X+SpringAMQP 整合RabbitMQ实战

第1集 SpringAMQP介绍+SpringBoot2.X项目 创建

简介：介绍SpringAMQP+创建SpringBoot2.X

- 什么是Spring-AMQP
 - 官网：<https://spring.io/projects/spring-amqp>
 - Spring 框架的AMQP消息解决方案，提供模板化的发送和接收消息的抽象层，提供基于消息驱动的 POJO的消息监听等
 - 提供不依赖于任何特定的AMQP代理实现或客户端库通用的抽象，最终用户代码将很容易实现更易替换、添加和删除AMQP，因为它可以只针对抽象层来开发
 - 总之就是提高我们的框架整合消息队列的效率，SpringBoot为更方便开发RabbitMQ推出了starter，

- 我们使用 spring-boot-starter-amqp 进行开发
- 创建新版SpringBoot2.X项目
 - 官网: <https://spring.io/projects/spring-boot>
 - 在线构建工具 <https://start.spring.io/>



- 如果版本或者链接找不到，可以直接导入课程项目
- 注意: 有些包maven下载慢，等待下载如果失败
 - 删除本地仓库spring相关的包，重新执行 mvn install
 - 建议先使用默认的maven仓库，不用更换地址
 - 基于当前项目仓库地址修改

```
<!-- 代码库 -->
<repositories>
  <repository>
    <id>maven-ali</id>
```

```
<url>http://maven.aliyun.com/nexus/content/groups/public//</url>
      <releases>
          <enabled>true</enabled>
      </releases>
      <snapshots>
          <enabled>true</enabled>

<updatePolicy>always</updatePolicy>

<checksumPolicy>fail</checksumPolicy>
      </snapshots>
  </repository>
</repositories>

<pluginRepositories>
  <pluginRepository>
    <id>public</id>
    <name>aliyun nexus</name>

<url>http://maven.aliyun.com/nexus/content/groups/public//</url>
      <releases>
          <enabled>true</enabled>
      </releases>
      <snapshots>
          <enabled>false</enabled>
      </snapshots>
```

```
</pluginRepository>  
</pluginRepositories>
```

- spring-boot-starter-amqp 依赖添加

```
<!--引入AMQP-->  
<dependency>  
  
<groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-  
amqp</artifactId>  
</dependency>
```

第2集 SpringBoot2.X整合RabbitMQ实战

简介：介绍SpringBoot2.X整合Rabbitmq实战

- yml配置文件修改

```
#消息队列
spring:
  rabbitmq:
    host: 10.211.55.13
    port: 5672
    virtual-host: /dev
    password: password
    username: admin
```

- RabbitMQConfig文件

```
@Configuration
public class RabbitMQConfig {

    public static final String EXCHANGE_NAME =
"order_exchange";
    public static final String QUEUE_NAME =
"order_queue";
    /**

```

```
* 交换机
* @return
*/
@Bean
public Exchange orderExchange() {
    return
ExchangeBuilder.topicExchange(EXCHANGE_NAME).du
rable(true).build();
    //return new
TopicExchange(EXCHANGE_NAME, true, false);
}

/**
 * 队列
 * @return
*/
@Bean
public Queue orderQueue() {
    return
QueueBuilder.durable(QUEUE_NAME).build();
    //return new Queue(QUEUE_NAME, true,
false, false, null);
}

/**
 * 交换机和队列绑定关系
*/
@Bean
```

```
    public Binding orderBinding(Queue queue,  
Exchange exchange) {  
    return  
        BindingBuilder.bind(queue).to(exchange).with("o  
rder.#").noargs();  
        //return new Binding(QUEUE_NAME,  
        Binding.DestinationType.QUEUE, EXCHANGE_NAME,  
        "order.#", null);  
    }  
}
```

- 消息生产者-测试类

```
@SpringBootTest
class DemoApplicationTests {

    @Autowired
    private RabbitTemplate template;

    @Test
    void send() {

        template.convertAndSend(RabbitMQConfig.EXCHANGE
                _NAME, "order.new", "新订单来啦1");
    }

}
```

- 消息消费者

```
@Component
@RabbitListener(queues = "order_queue")
public class OrderMQListener {

    /**
     * RabbitHandler 会自动匹配 消息类型（消息自动确认）
     * @param msg
     * @param message
    }
```

```
* @throws IOException
*/
@RabbitHandler
public void releaseCouponRecord(String msg,
Message message) throws IOException {

    long msgTag =
message.getMessageProperties().getDeliveryTag()
;

    System.out.println("msgTag="+msgTag);

System.out.println("message="+message.toString());
}

System.out.println("监听到消息：消息内
容："+message.getBody());
}

}
```



愿景："让编程不再难学，让技术与生活更加有趣"

第十章 高级特性-RabbitMQ消息可靠性投递+消费

第1集 高级知识点之RabbitMQ消息可靠性投递讲解

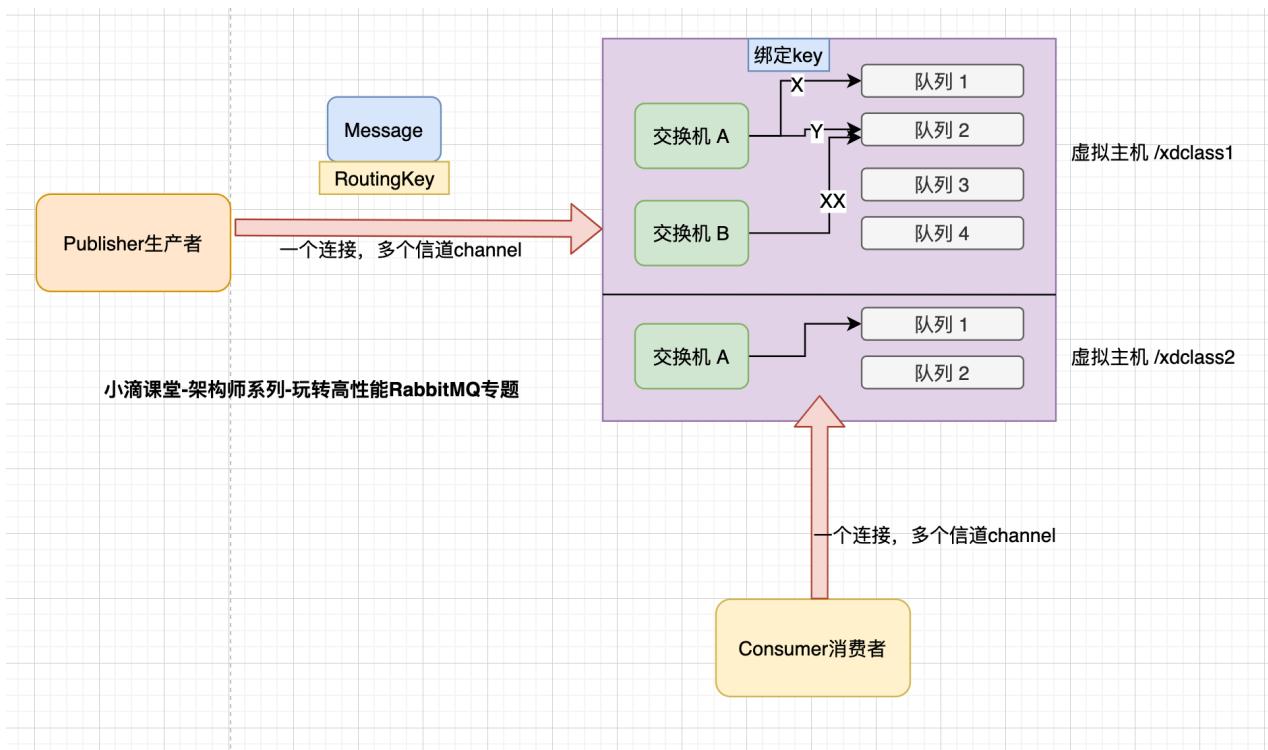
简介：Rabbitmq的消息可靠性投递讲解

- 什么是消息的可靠性投递
 - 保证消息百分百发送到消息队列中去
 - 详细

- 保证mq节点成功接受消息
- 消息发送端需要接受到mq服务端接受到消息的确认应答
- 完善的消息补偿机制，发送失败的消息可以再感知并二次处理



- RabbitMQ消息投递路径
 - 生产者-->交换机->队列->消费者
 - 通过两个的点控制消息的可靠性投递
 - 生产者到交换机
 - 通过confirmCallback
 - 交换机到队列
 - 通过returnCallback
- 建议
 - 开启消息确认机制以后，保证了消息的准确送达，但由于频繁的确认交互，rabbitmq 整体效率变低，吞吐量下降严重，不是非常重要的消息真心不建议用消息确认机制



第2集 新版RabbitMQ消息可靠性投递 confirmCallback实战

简介： Rabbitmq的消息可靠性投递confirmCallback实战

- 生产者到交换机
 - 通过confirmCallback
 - 生产者投递消息后，如果Broker收到消息后，会给生产者一个ACK。生产者通过ACK，可以确认这条消息是否正常发送到Broker，这种方式是消息可靠性投递的核心
- 开启confirmCallback

```
#旧版，确认消息发送成功，通过实现ConfirmCallBack接口，消息发送到交换器Exchange后触发回调
```

```
spring.rabbitmq.publisher-confirms=true
```

```
#新版，NONE值是禁用发布确认模式，是默认值，  
CORRELATED值是发布消息成功到交换器后会触发回调方法
```

```
spring.rabbitmq.publisher-confirm-type:  
correlated
```

- 开发实战

```
@Autowired  
private RabbitTemplate template;
```

```
@Test  
void testConfirmCallback() {
```

```
        template.setConfirmCallback(new
RabbitTemplate.ConfirmCallback() {
    /**
     *
     * @param correlationData 配置
     * @param ack 交换机是否收到消息, true是成
功, false是失败
     * @param cause 失败的原因
    */
    @Override
    public void confirm(CorrelationData
correlationData, boolean ack, String cause) {
        System.out.println("confirm=====>");
        System.out.println("confirm====
ack="+ack);
        System.out.println("confirm====
cause="+cause);

        //根据ACK状态做对应的消息更新操作 TODO
    }
});

template.convertAndSend(RabbitMQConfig.EXCHAN
GE_NAME+, "order.new", "新订单来啦1");
}
```

- 模拟异常：修改投递的交换机名称

第3集 新版RabbitMQ消息可靠性投递 returnCallback实战

简介： Rabbitmq的消息可靠性投递returnCallback实战

- 交换机到队列
 - 通过returnCallback
 - 消息从交换器发送到对应队列失败时触发
 - 两种模式
 - 交换机到队列不成功，则丢弃消息（默认）
 - 交换机到队列不成功，返回给消息生产者，触发

returnCallback

```
//为true，则交换机处理消息到路由失败，则会返回给生产者  
//或者配置文件  
spring.rabbitmq.template.mandatory=true  
template.setMandatory(true);
```

- 第一步 开启returnCallback配置

#新版

```
spring.rabbitmq.publisher-returns=true
```

- 第二步 修改交换机投递到队列失败的策略

```
#为true，则交换机处理消息到路由失败，则会返回给生产者  
spring.rabbitmq.template.mandatory=true
```

- 开发实战

```
@Test  
void testReturnCallback() {  
    //为true，则交换机处理消息到路由失败，则会返回给生产者  
    //开启强制消息投递（mandatory为设置为true），但消息未被路由至任何一个queue，则回退一条消息
```

```
template.setReturnsCallback(new
RabbitTemplate.ReturnsCallback() {
    @Override
    public void
returnedMessage(ReturnedMessage returned) {
        int code = returned.getReplyCode();
        System.out.println("code="+code);

System.out.println("returned="+returned.toStrin
g());
    }
} );

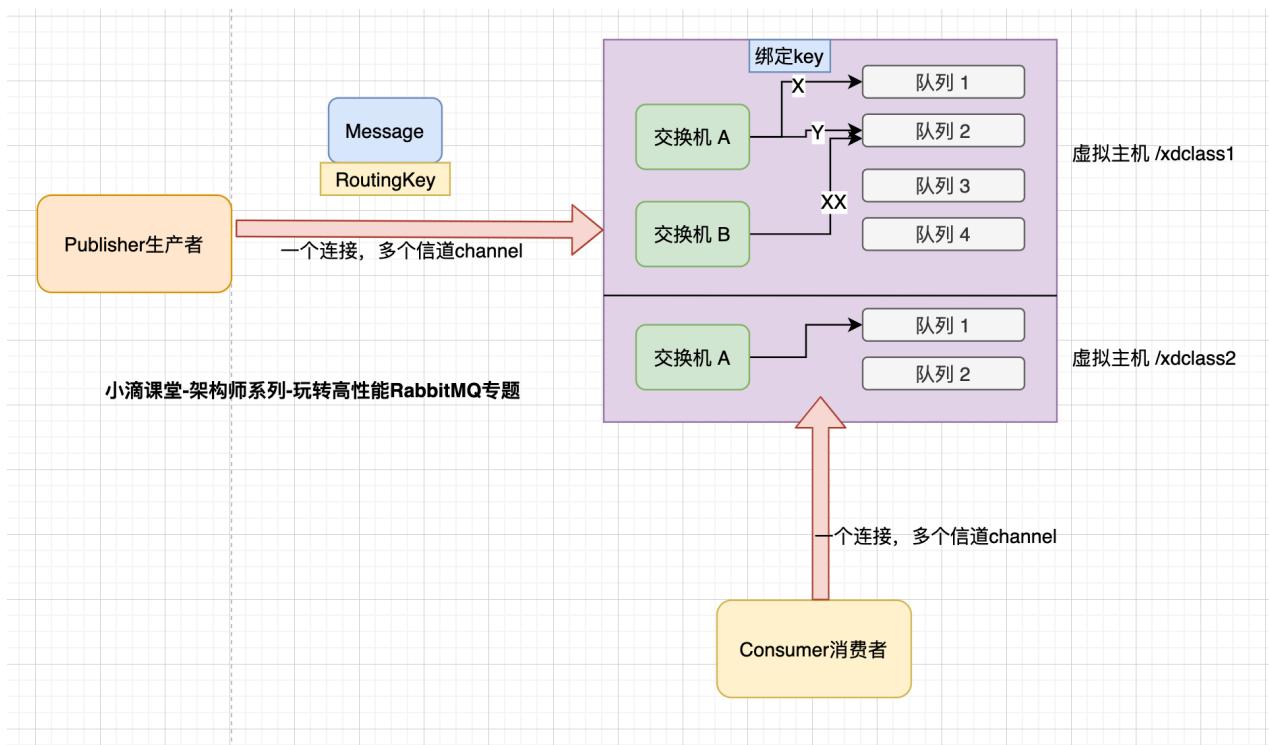
template.convertAndSend(RabbitMQConfig.EXCHANGE
_NAME, "xxx.order.new", "新订单来啦11");
}
```

- 模拟异常，修改路由key,拼接不存在的路由

第4集 高级特性之RabbitMQ消息确认机制ACK讲解

简介： Rabbitmq的消息确机制ACK讲解

- 背景：消费者从broker中监听消息，需要确保消息被合理处理



- RabbitMQ的ACK介绍
 - 消费者从RabbitMQ收到消息并处理完成后，反馈给 RabbitMQ， RabbitMQ收到反馈后才将此消息从队列中删除
 - 消费者在处理消息出现了网络不稳定、服务器异常等现象，那么就不会有ACK反馈， RabbitMQ会认为这个消息没有正常消费，会将消息重新放入队列中
 - 只有当消费者正确发送ACK反馈， RabbitMQ确认收到后，消息才会从RabbitMQ服务器的数据中删除。
 - 消息的ACK确认机制默认是打开的，消息如未被进行 ACK的消息确认机制，这条消息被锁定Unacked
- 确认方式
 - 自动确认（默认）
 - 手动确认 manual

```
spring:  
    rabbitmq:  
        #开启手动确认消息，如果消息重新入队，进行重试  
        listener:  
            simple:  
                acknowledge-mode: manual
```

- 其他（基本不用，忽略）

第5集 RabbitMQ消息确认机制ACK配置实战 +DeliveryTag+Reject介绍

简介： Rabbitmq的消息确机制ACK实战+DeliveryTag介绍

- 代码实战

```

@RabbitHandler
public void releaseCouponRecord(String body,
Message message, Channel channel) throws
IOException {
    long msgTag =
message.getMessageProperties().getDeliveryTag
();
    System.out.println("msgTag="+msgTag);

System.out.println("message="+message.toString());
    System.out.println("body="+body);

    //成功确认，使用此回执方法后，消息会被
    //rabbitmq broker 删除
    //channel.basicAck(msgTag, false);

    //channel.basicNack(msgTag, false, true);

}

```

- deliveryTag介绍
- 表示消息投递序号，每次消费消息或者消息重新投递后，deliveryTag都会增加
- basicNack和basicReject介绍
 - basicReject一次只能拒绝接收一个消息，可以设置是否 requeue。

- basicNack方法可以支持一次0个或多个消息的拒收，可以设置是否requeue。
- 人工审核异常消息
 - 设置重试阈值，超过后确认消费成功，记录消息，人工处理



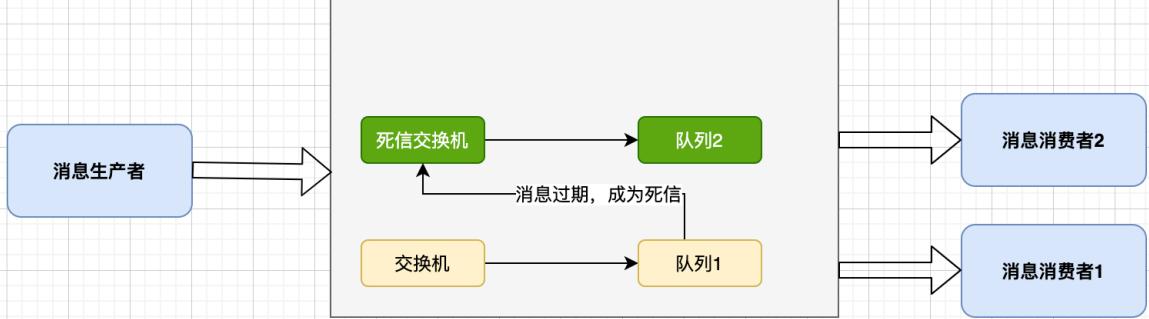
愿景："让编程不再难学，让技术与生活更加有趣"

第十一章 高级特性-RabbitMQ TTL死信队列+延迟队列 实战

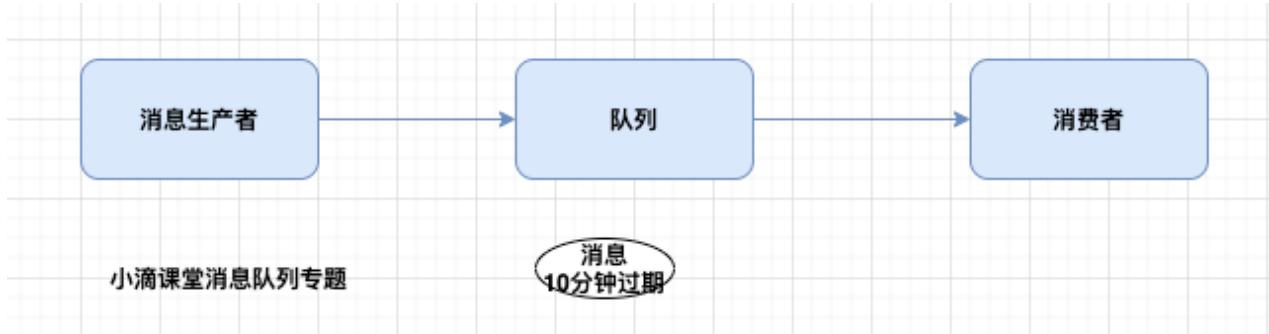
第1集 高级特性之-RabbitMQ死信队列 + TTL介绍《上》

简介：讲解RabbitMQ的的死信队列+ TTL 《上》

- 什么是TTL
 - time to live 消息存活时间
 - 如果消息在存活时间内未被消费，则会别清除
 - RabbitMQ支持两种ttl设置
 - 单独消息进行配置ttl
 - 整个队列进行配置ttl（居多）
- 什么是rabbitmq的死信队列
 - 没有被及时消费的消息存放的队列
- 什么是rabbitmq的死信交换机
 - Dead Letter Exchange (死信交换机，缩写：DLX) 当消息成为死信后，会被重新发送到另一个交换机，这个交换机就是DLX死信交换机。



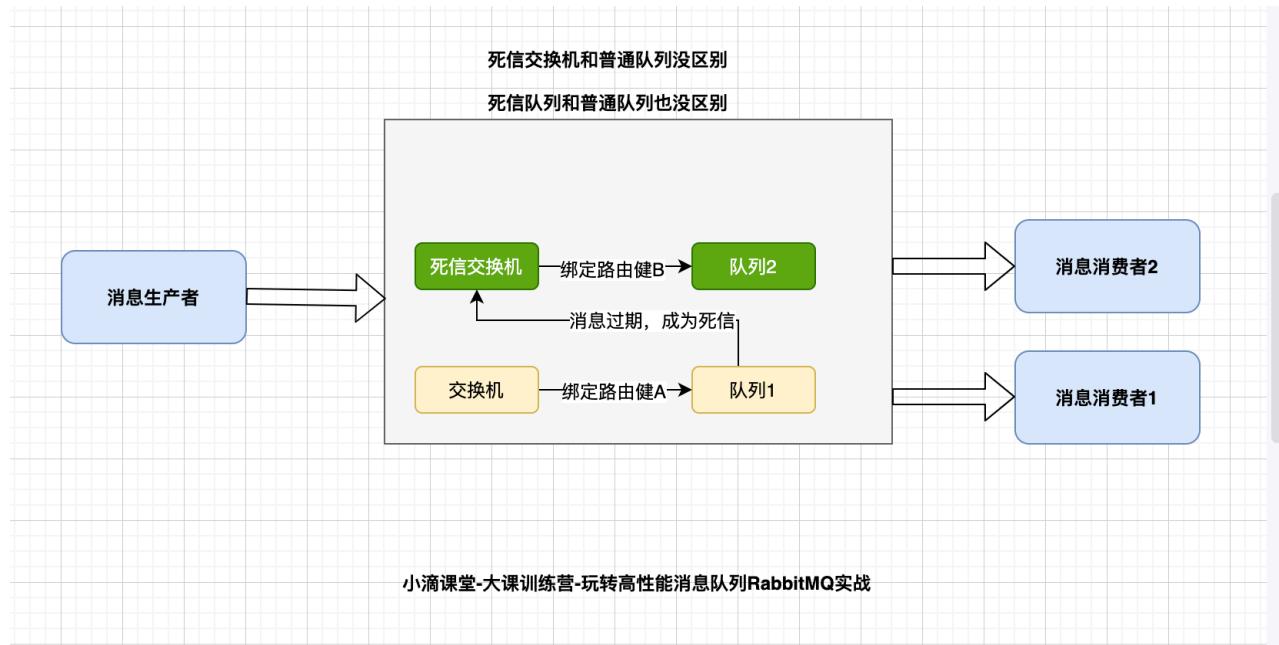
小滴课堂-大课训练营-玩转高性能消息队列RabbitMQ实战



- 消息有哪几种情况成为死信
 - 消费者拒收消息 (**basic.reject/ basic.nack**) , 并且没有重新入队 **requeue=false**
 - 消息在队列中未被消费, 且超过队列或者消息本身过期时间**TTL(time-to-live)**
 - 队列的消息长度达到极限
 - 结果: 消息成为死信后, 如果该队列绑定了死信交换机, 则消息会被死信交换机重新路由到死信队列

第2集 高级特性之-RabbitMQ死信队列 + TTL介绍《下》

简介：讲解RabbitMQ的死信队列+ TTL《下》



- RabbitMQ 管控台消息TTL测试
 - 队列过期时间使用参数，对整个队列消息统一过期
 - x-message-ttl
 - 单位ms(毫秒)
 - 消息过期时间使用参数（如果队列头部消息未过期，队列中级消息已经过期，已经还在队列里面）
 - expiration
 - 单位ms(毫秒)
 - 两者都配置的话，时间短的先触发
- RabbitMQ Web控制台测试
 - 新建死信交换机(和普通没区别)

▼ Add a new exchange

Virtual host:	/test
Name:	dead_exchange_1 *
Type:	topic
Durability:	Durable
Auto delete:	No
Internal:	No
Arguments:	<input type="text"/> = <input type="text"/> String
Add Alternate exchange ?	

Add exchange

- 新建死信队列 (和普通没区别)

▼ Add a new queue

Virtual host:	/test
Type:	Classic
Name:	dead_queue *
Durability:	Durable
Auto delete:	No
Arguments:	= String

Add [Message TTL](#) | [Auto expire](#) | [Max length](#) | [Max length bytes](#) | [Overflow behaviour](#) | [Dead letter exchange](#) | [Dead letter routing key](#) | [Single active consumer](#) | [Maximum priority](#) | [Lazy mode](#) | [Master locator](#)

Add queue

- 死信交换机和队列绑定

Add binding from this exchange

To queue	dead_queue *
Routing key:	dead.#
Arguments:	= String

Bind

- 新建普通队列，设置过期时间、指定死信交换机

▼ Add a new queue

Virtual host:	/dev
Type:	Classic
Name:	product_queue *
Durability:	Durable
Auto delete:	No
Arguments:	x-message-ttl = 10000 Number x-dead-letter-exchange = dead_exchange String x-dead-letter-routing-key = dead.product String = String

Add [Message TTL](#) | [Auto expire](#) | [Max length](#) | [Max length bytes](#) | [Overflow behaviour](#) | [Dead letter exchange](#) | [Dead letter routing key](#) | [Single active consumer](#) | [Maximum priority](#) | [Lazy mode](#) | [Master locator](#)

Add queue

- 测试：直接web控制台往product_queue发送消息即可

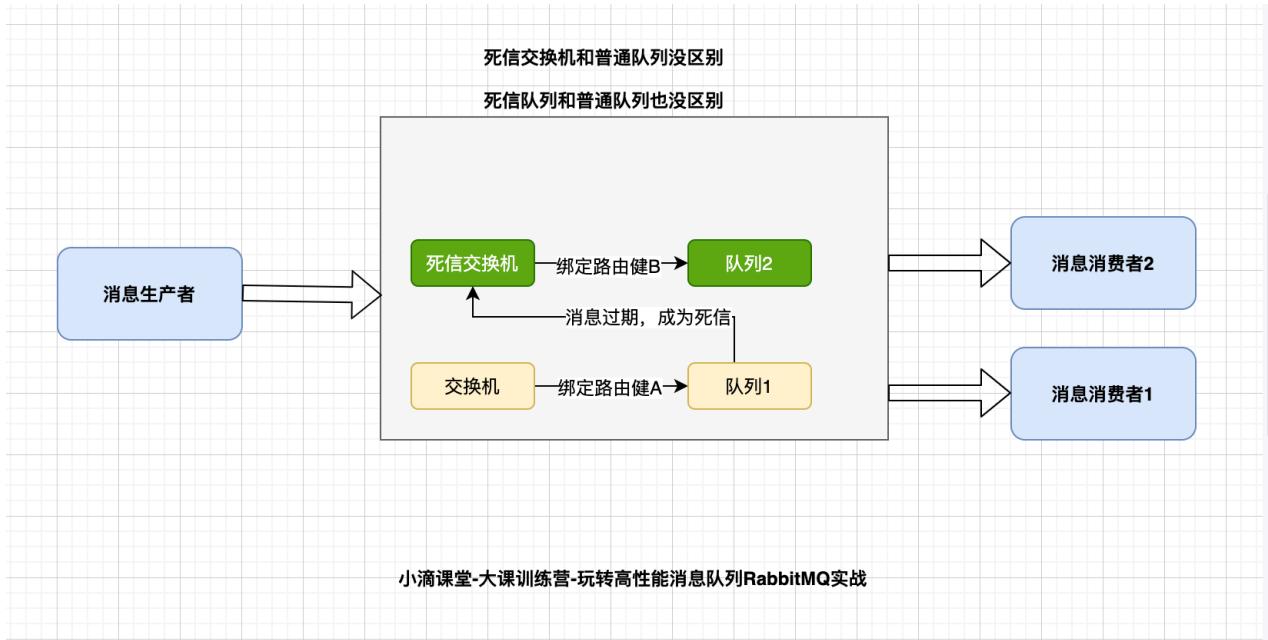
第3集 高级特性之-RabbitMQ 延迟队列介绍和应用场景

简介：讲解RabbitMQ的延迟队列和应用场景

- 什么是延迟队列
 - 一种带有延迟功能的消息队列，Producer 将消息发送到消息队列 服务端，但并不期望这条消息立马投递，而是推送到在当前时间点之后的某一个时间投递到 Consumer 进行消费，该消息即定时消息
- 使用场景
 - 通过消息触发一些定时任务，比如在某一固定时间点向用户发送提醒消息
 - 用户登录之后5分钟给用户做分类推送、用户多少天未登录给用户做召回推送；
 - 消息生产和消费有时间窗口要求：比如在天猫电商交易中超时未支付关闭订单的场景，在订单创建时会发送一

条 延时消息。这条消息将会在 30 分钟以后投递给消费者，消费者收到此消息后需要判断对应的订单是否已完成支付。如支付未完成，则关闭订单。如已完成支付则忽略

- Cloud微服务大课训练营里面的应用
 - 优惠券回收
 - 商品库存回收
- 业界的一些实现方式
 - 定时任务高精度轮训
 - 采用RocketMQ自带延迟消息功能
 - RabbitMQ本身是不支持延迟队列的，怎么办?
 - 结合死信队列的特性，就可以做到延迟消息





小滴课堂 愿景："让编程不再难学，让技术与生活更加有趣"

第十二章 综合实战-主题模式+延迟消息-实现新商家规定时间内上架商品检查

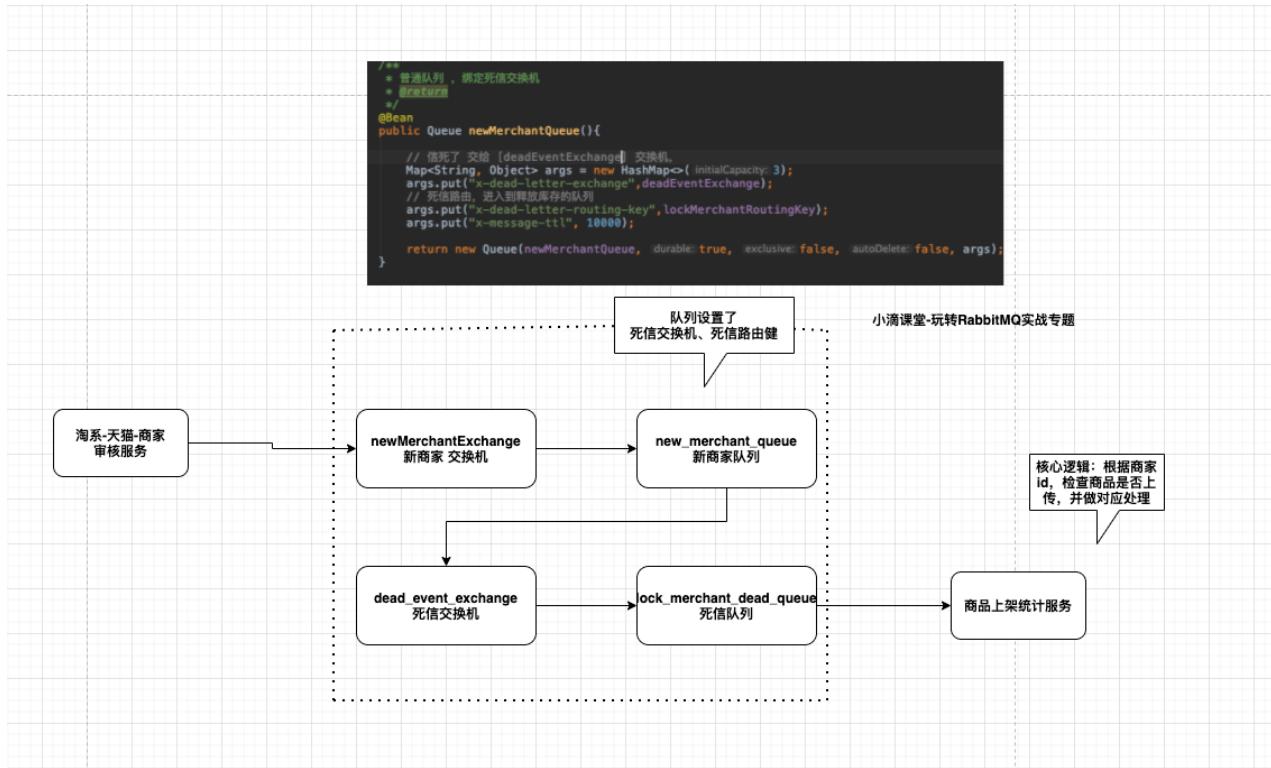
第1集 案例实战-延迟消息实现业务逻辑介绍

简介：讲解RabbitMQ的案例实战业务逻辑介绍

- 背景

JD、淘系、天猫、拼多多电商平台，规定新注册的商家，审核通过后需要在【规定时间】内上架商品，否则冻结账号。

- 使用RabbitMQ实现



第2集 案例实战-SpringBoot2.X+RabbitMQ延迟消息配置开发

简介：讲解RabbitMQ的案例实战配置开发

- 死信交换机和死信队列开发

```
/**
 * 死信队列
 */
public static final String
LOCK_MERCHANT_DEAD_QUEUE =
"lock_merchant_dead_queue";

/**
 * 死信交换机
 */
public static final String
LOCK_MERCHANT_DEAD_EXCHANGE =
"lock_merchant_dead_exchange";

/**
 * 进入死信队列的路由key
 */
public static final String
LOCK_MERCHANT_ROUTING_KEY =
"lock_merchant_routing_key";

/**
 * 创建死信交换机
 * @return
 */
@Bean
public Exchange lockMerchantDeadExchange() {
```

```
        return new
TopicExchange(LOCK_MERCHANT_DEAD_EXCHANGE,true,
false);
    }

/**
 * 创建死信队列
 * @return
 */
@Bean
public Queue lockMerchantDeadQueue(){
    return
QueueBuilder.durable(LOCK_MERCHANT_DEAD_QUEUE).
build();
}

/**
 * 绑定死信交换机和死信队列
 * @return
*/
@Bean
public Binding lockMerchantBinding(){

    return new
Binding(LOCK_MERCHANT_DEAD_QUEUE,Binding.Destin
ationType.QUEUE,
LOCK_MERCHANT_DEAD_EXCHANGE,LOCK_MERCHANT_ROUTI
NG_KEY,null);
}
```

```
}
```

- topic交换机和队列开发, 绑定死信交换机

```
/***
 * 普通队列, 绑定的个死信交换机
 */
public static final String
NEW_MERCHANT_QUEUE = "new_merchant_queue";

/***
 * 普通的topic交换机
 */
public static final String
NEW_MERCHANT_EXCHANGE =
"new_merchant_exchange";

/***
 * 路由key
 */
public static final String
NEW_MERCHANT_ROUTIING_KEY =
"new_merchant_routing_key";

/***
 * 创建普通交换机
 * @return

```

```
 */
@Bean
public Exchange newMerchantExchange(){
    return new
TopicExchange(NEW_MERCHANT_EXCHANGE,true,false)
;
}

/**
 * 创建普通队列
 * @return
 */
@Bean
public Queue newMerchantQueue(){

    Map<String, Object> args = new HashMap<>
(3);
    //消息过期后，进入到死信交换机
    args.put("x-dead-letter-
exchange",LOCK_MERCHANT_DEAD_EXCHANGE);

    //消息过期后，进入到死信交换机的路由key
    args.put("x-dead-letter-routing-
key",LOCK_MERCHANT_ROUTING_KEY);

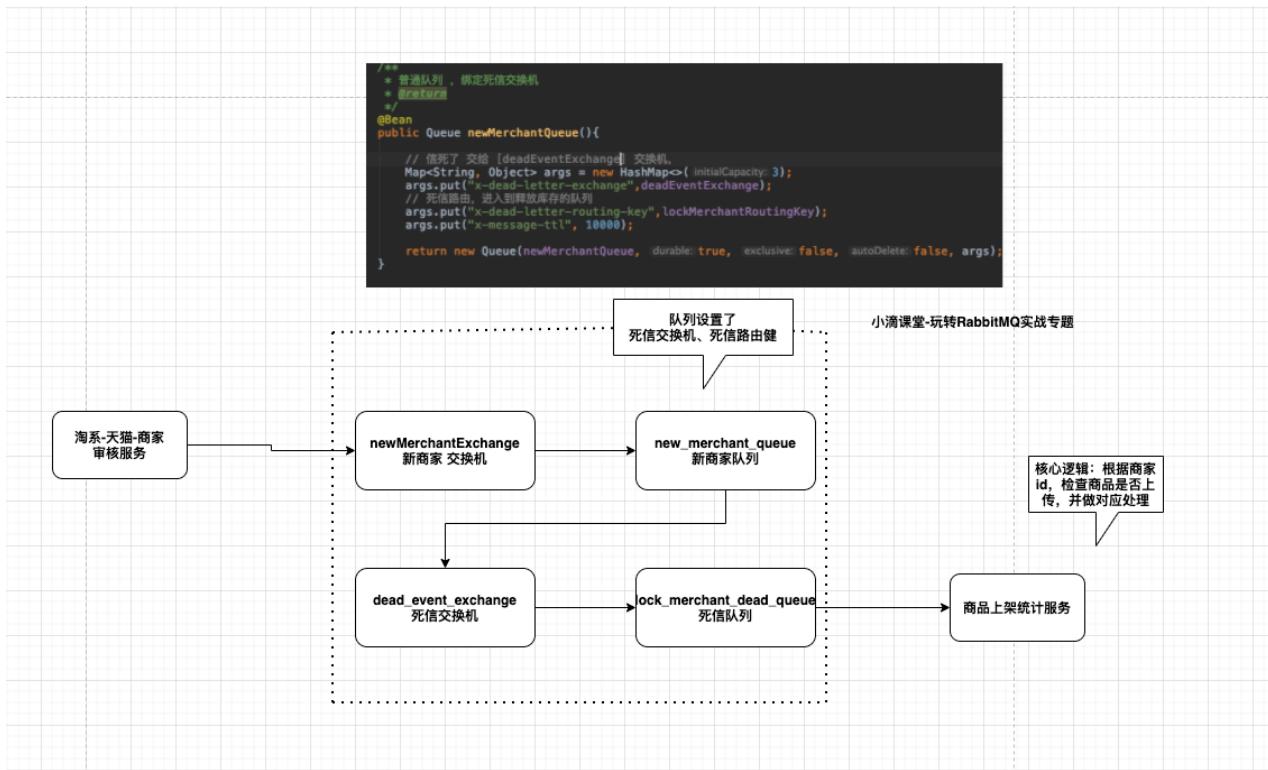
    //过期时间，单位毫秒
    args.put("x-message-ttl",10000);
```

```
        return  
QueueBuilder.durable(NEW_MERCHANT_QUEUE).withAr  
guments(args).build();  
  
    }  
  
    /**  
     * 绑定交换机和队列  
     * @return  
     */  
    @Bean  
    public Binding newMerchantBinding(){  
  
        return new  
Binding(NEW_MERCHANT_QUEUE,Binding.DestinationT  
ype.QUEUE,  
NEW_MERCHANT_EXCHANGE,NEW_MERCHANT_ROUTIING_KEY  
,null);  
    }  
}
```

第3集 案例实战-SpringBoot2.X 延迟消息生产和消费实战开发

简介：讲解RabbitMQ的案例实战-消息生产和消费

- 消息生产
 - 投递到普通的topic交换机
 - 消息过期，进入死信交换机
- 消息消费
 - 消费者监听死信交换机的队列



 小滴课堂 愿景：“让编程不再难学，让技术与生活更加有趣”

第十三章 玩转高性能消息队列 RabbitMQ课程总结和后续路线

第1集 玩转高性能消息队列RabbitMQ课程总结

简介：玩转高性能消息队列RabbitMQ课程总结

- 为什么要用MQ，JMS和AMQP的区别
- RabbitMQ常见概念和基础模型
- Docker快速入门
- 掌握多种RabbitMQ工作模式
- 可靠性投递和消费、死信队列、延迟队列
- 课程总结

我信你个鬼 你个糟老头子



- 加餐内容
 - 高可用集群实战+SpringBoot整合
 - RabbitMQ大厂里面的面试题+原理



加油，你是最胖的

第2集 成长必备- 如何选择IT技术人的职场充电站

简介：小滴课堂架构师学习路线和大课训练营介绍

- 小滴课堂永久会员 & 小滴课堂年度会员

- 适合IT后端人员，零基础就业、在职充电、架构师学习路线-专题技术方向
 - 包括业界主流技术栈，前端、后端、测试、架构等课程，接近上百套视频，深度+广度
 - 包括从零基础到就业班，高级工程师、技术负责人、架构师技术栈全套学习路线
 - 永久会员可以观看全部专题IT技术，作为一个终生学习平台，每个月更新多套视频
 - 如何获取最新路线图，有你想学的多数主流技术栈，持续更新！！！



可別枯了

- 大课综合训练营
 - 适合用于专项拔高，适合有一定工作经验的同学，架构师方向发展的训练营，包括多个方向
 - 综合项目大课训练营
 - 海量数据分库分表大课训练营
 - 架构解决方案大课训练营
 - 全链路性能优化大课训练营
 - 安全攻防大课训练营
 - 数据分析大课训练营
 - 算法刷题大课训练营
- 直播小班课（留意官网即可或者联系我即可）
- 技术人的导航站（涵盖我们主流的技术网站、社区、工具等等，即将上线）
 - 地址：open1024.com



愿景：“让编程不再难学，让技术与生活更加有趣”

第十四章 高级特性-Rabbitmq高可用集群实战讲解

第1集 RabbitMQ高可用普通集群模式介绍

简介：讲解RabbitMQ高可用普通集群模式介绍

- 背景
 - 掌握了消息的可靠性投递，还有消费，假如mq节点宕机了怎么办？
 - 因此需要做多节点集群配置

Global counts ?									+/-	
Connections: 1		Channels: 1		Exchanges: 27		Queues: 4		Consumers: 1		
▼ Nodes										
Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats	+/-	
rabbit@rabbit_host1	43 1048576 available	1 943629 available	591 1048576 available	110 MiB 733 MiB high watermark& 88 MiB low watermark	44 GiB	8h 58m	basic disc 2 rss	This node All nodes		

- RabbitMQ集群模式一介绍

- 普通集群

默认的集群模式，比如有节点 node1和node2、node3，三个节点是普通集群，但是他们仅有相同的元数据，即交换机、队列的结构；

案例：

消息只存在其中的一个节点里面，假如消息A，存储在node1节点，

消费者连接node1个节点消费消息时，可以直接取出来；

但如果 消费者是连接的是其他节点

那rabbitmq会把 queue 中的消息从存储它的节点中取出，并经过连接节点转发后再发送给消费者

问题：

假如node1故障，那node2无法获取node1存储未被消费的消息；

如果node1持久化后故障，那需要等node1恢复后才可以正常消费

如果node1没做持久化后故障，那消息将会丢失

这个情况无法实现高可用性，且节点间会增加通讯获取消息，性能存在瓶颈

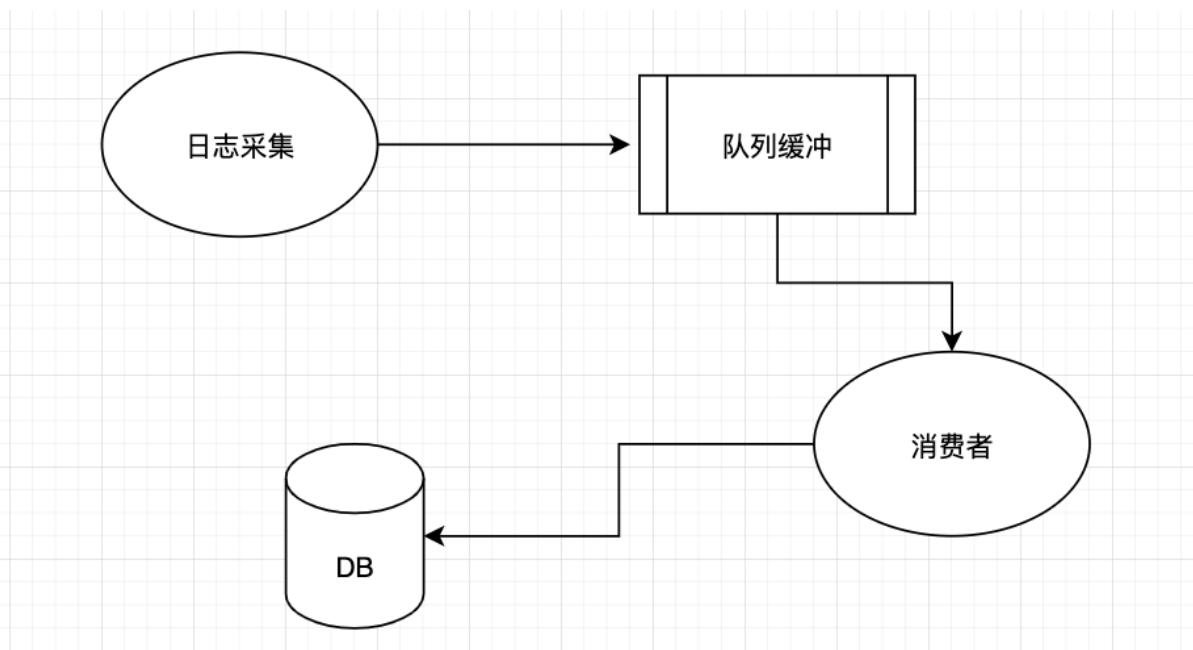
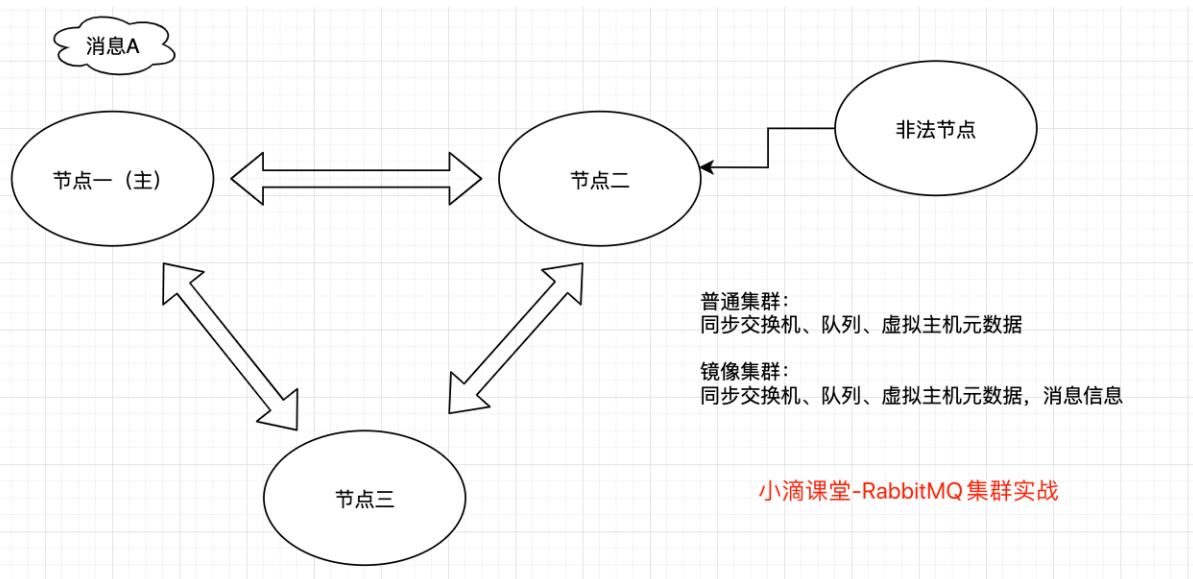
项目中springboot+amqp里面需要写多个节点的配置，比如下面

```
spring.rabbitmq.addresses =  
192.168.1.1:5672,192.168.1.2:5672,192.168.1.3  
:5672
```

该模式更适合于消息无需持久化的场景，如日志传输的队列

- 注意：集群需要保证各个节点有相同的token令牌

`erlang.cookie`是erlang的分布式token文件，集群内各个节点的`erlang.cookie`需要相同，才可以互相通信



第2集 RabbitMQ高可用mirror镜像集群模式介绍

简介：讲解RabbitMQ高可用mirror镜像集群模式介绍

- 镜像集群（大厂基本使用这种方式）

队列做成镜像队列，让各队列存在于多个节点中
和普通集群比较大的区别就是【队列queue的消息message】
会在集群各节点之间同步，且并不是在 consumer 获取数据时
临时拉取，而普通集群则是临时从存储的节点里面拉取对应的数据
据

结论：

实现了高可用性，部分节点挂掉后，不影响正常的消费
可以保证100%消息不丢失，推荐3个奇数节点，结合
LVS+Keepalive进行IP漂移，防止单点故障

缺点：由于镜像队列模式下，消息数量过多，大量的消息同步也
会加大网络带宽开销，适合高可用要求比较高的项目
过多节点的话，性能则更加受影响

- 注意：集群需要保证各个节点有相同的token令牌

`erlang.cookie`是erlang的分布式token文件，集群内各个节
点的`erlang.cookie`需要相同，才可以互相通信

- 还有其他通过插件形成的集群，比如Federation集群，大家
有兴趣可以去了解下

第3集 综合实战-RabbitMQ高可用普通集群 搭建基础准备

简介：讲解RabbitMQ高可用普通集群搭建基础准备

- 清理单机和网络开发
 - 关闭原先的单节点
 - 阿里网络安全组开放对应的端口
 - 防火墙一定要关闭
- 准备3个节点安装好rabbitmq，形成集群（记得每个节点间隔几十秒再启动，如果失败删除宿主机文件重新搭建）

```
#节点一，主节点，创建-v映射目录
```

```
docker run -d --hostname rabbit_host1 --name
rabbitmq1 -p 15672:15672 -p 5672:5672 -e
RABBITMQ_NODENAME=rabbit -e
RABBITMQ_DEFAULT_USER=admin -e
RABBITMQ_DEFAULT_PASS=xdclass.net168 -e
RABBITMQ_ERLANG_COOKIE='rabbitmq_cookie_xdclass
' --privileged=true -v
/usr/local/rabbitmq/1/lib:/var/lib/rabbitmq -v
/usr/local/rabbitmq/1/log:/var/log/rabbitmq
rabbitmq:management
```

#节点二,创建-v映射目录

```
docker run -d --hostname rabbit_host2 --name
rabbitmq2 -p 15673:15672 -p 5673:5672 --link
rabbitmq1:rabbit_host1 -e
RABBITMQ_NODENAME=rabbit -e
RABBITMQ_DEFAULT_USER=admin -e
RABBITMQ_DEFAULT_PASS=xdclass.net168 -e
RABBITMQ_ERLANG_COOKIE='rabbitmq_cookie_xdclass
' --privileged=true -v
/usr/local/rabbitmq/2/lib:/var/lib/rabbitmq -v
/usr/local/rabbitmq/2/log:/var/log/rabbitmq
rabbitmq:management
```

#节点三,创建-v映射目录

```
docker run -d --hostname rabbit_host3 --name
rabbitmq3 -p 15674:15672 -p 5674:5672 --link
rabbitmq1:rabbit_host1 --link
rabbitmq2:rabbit_host2 -e
RABBITMQ_NODENAME=rabbit -e
RABBITMQ_DEFAULT_USER=admin -e
RABBITMQ_DEFAULT_PASS=xdclass.net168 -e
RABBITMQ_ERLANG_COOKIE='rabbitmq_cookie_xdclass
' --privileged=true -v
/usr/local/rabbitmq/3/lib:/var/lib/rabbitmq -v
/usr/local/rabbitmq/3/log:/var/log/rabbitmq
rabbitmq:management
```

- 参数说明

--hostname 自定义Docker容器的 hostname

--link 容器之间连接, link不可或缺, 使得三个容器能互相通信

--privileged=true 使用该参数, container内的root拥有真正的root权限, 否则容器出现permission denied

-v 宿主机和容器路径映射

参数 RABBITMQ_NODENAME, 缺省 Unix*:

rabbit@\$HOSTNAME

参数 RABBITMQ_DEFAULT_USER=admin

参数 RABBITMQ_DEFAULT_PASS=xdclass.net168

Erlang Cookie 值必须相同, 也就是一个集群内 RABBITMQ_ERLANG_COOKIE 参数的值必须相同, 相当于不同节点之间通讯的密钥, erlang.cookie是erlang的分布式 token文件, 集群内各个节点的erlang.cookie需要相同, 才可以互相通信

第4集 综合实战-RabbitMQ高可用普通集群搭建

简介：讲解RabbitMQ高可用普通集群搭建

- 配置集群

节点一配置集群

```
docker exec -it rabbitmq1 bash  
rabbitmqctl stop_app  
rabbitmqctl reset  
rabbitmqctl start_app  
exit
```

节点二加入集群，--ram是以内存方式加入，忽略该参数默认为磁盘节点。

```
docker exec -it rabbitmq2 bash  
rabbitmqctl stop_app
```

```
rabbitmqctl join_cluster --ram  
rabbit@rabbit_host1  
rabbitmqctl start_app  
exit
```

节点三加入集群，`--ram`是以内存方式加入，忽略该参数默认为磁盘节点。

```
docker exec -it rabbitmq3 bash  
rabbitmqctl stop_app  
rabbitmqctl reset  
rabbitmqctl join_cluster --ram  
rabbit@rabbit_host1  
rabbitmqctl start_app  
exit
```

#查看集群节点状态，配置启动了3个节点，1个磁盘节点和2个内存节点

```
rabbitmqctl cluster_status
```

```

1. root@centos-linux:~ (ssh)
[root@centos-linux ~]# docker exec -it rabbitmq3 bash
root@rabbit_host3:/# rabbitmqctl cluster_status
RABBITMQ_ERLANG_COOKIE env variable support is deprecated and will be REMOVED in a future version. Use the $HOME/.erlang.cookie file or the --erlang-cookie switch instead.
Cluster status of node rabbit@rabbit_host3 ...
Basics
Cluster name: rabbit@rabbit_host1
Disk Nodes
rabbit@rabbit_host1
RAM Nodes
rabbit@rabbit_host2
rabbit@rabbit_host3
Running Nodes
rabbit@rabbit_host1
rabbit@rabbit_host2
rabbit@rabbit_host3
Versions
rabbit@rabbit_host1: RabbitMQ 3.8.9 on Erlang 23.2.1
rabbit@rabbit_host2: RabbitMQ 3.8.9 on Erlang 23.2.1
rabbit@rabbit_host3: RabbitMQ 3.8.9 on Erlang 23.2.1
Maintenance status
Node: rabbit@rabbit_host1, status: not under maintenance
Node: rabbit@rabbit_host2, status: not under maintenance
Node: rabbit@rabbit_host3, status: not under maintenance
Alarms
(none)
Network Partitions
(none)
Listeners
Node: rabbit@rabbit_host1, interface: [::], port: 25672, protocol: clustering, purpose: inter-node and CLI tool communication

```

- 访问节点一的web管控台，可以看到多个节点

The screenshot shows the RabbitMQ Management UI's Overview page. At the top, there are tabs for Overview, Connections, Channels, Exchanges, Queues, and Admin. The Overview tab is selected. Below the tabs, there are two sections: 'Totals' and 'Nodes'.

Totals:

- File descriptors: 37
- Socket descriptors: 0
- Erlang processes: 567
- Memory: 110 MiB (green)
- Disk space: 43 GiB (green)
- Uptime: 6h 41m
- Info: basic disc 2 rss
- Reset stats: This node, All nodes

Nodes:

Name	File descriptors	Socket descriptors	Erlang processes	Memory	Disk space	Uptime	Info	Reset stats
rabbit@rabbit_host1	37	0	567	110 MiB	43 GiB	6h 41m	basic disc 2 rss	This node, All nodes
rabbit@rabbit_host2	35	0	563	108 MiB	43 GiB	6h 41m	basic RAM 2 rss	This node, All nodes
rabbit@rabbit_host3	36	0	564	103 MiB	43 GiB	6h 55m	basic RAM 2 rss	This node, All nodes

- 到此为止，我们已经完成了RabbitMQ普通模式集群的建立，启动了3个节点，1个磁盘节点和2个内存节点
- 测试
 - node1 主节点创建队列，发送消息 (可以选择消息是否持久化)
 - node2和node3通过节点自身的web管控台可以看到队列和消息

- 问题：如果把node1节点停止， node2和node3会收不到消息
- 备注：如果是在非主节点(非磁盘节点)创建队列和发送消息，则其他队列可以显示

第5集 综合实战-RabbitMQ高可用普通集群 SpringBoot测试

简介：讲解RabbitMQ高可用普通集群项目配置

- SpringBoot项目配置集群

```
#配置文件修改
#消息队列
spring:
    rabbitmq:
        addresses:
            10.211.55.13:5672,10.211.55.13:5673,10.211.55.1
            3:5674
        virtual-host: /dev
        password: xdclass.net168
```

```
username: admin
#开启消息二次确认,生产者到broker的交换机
publisher-confirm-type: correlated

#开启消息二次确认, 交换机到队列的可靠性投递
publisher-returns: true
#为true,则交换机处理消息到路由失败, 则会返回给生产
者
template:
    mandatory: true

#消息手工确认ACK
listener:
    simple:
        acknowledge-mode: manual
```

- 其他不用变动，正常的发消息
 - 模拟异常
 - 先发送消息，然后停止node1，启动SpringBoot项目，消息则消费不了，且报错
 - 重新启动node1，启动SpringBoot项目，恢复正常

第6集 综合实战-RabbitMQ高可用mirror镜像集群策略配置

简介：讲解RabbitMQ高可用mirror镜像集群配置策略配置

- 背景
 - 前面搭建了普通集群，如果磁盘节点挂掉后，如果没开启持久化数据就丢失了，其他节点也无法获取消息，所以我们这个集群方案需要进一步改造为**镜像模式集群**。
- 策略policy介绍

rabbitmq的策略policy是用来控制和修改集群的vhost队列和Exchange复制行为

就是要设置哪些Exchange或者queue的数据需要复制、同步，以及如何复制同步

- 创建一个策略来匹配队列

- 路径： rabbitmq管理页面 → Admin → Policies → Add / update a policy
- 参数： 策略会同步同一个VirtualHost中的交换器和队列数据
 - name： 自定义策略名称
 - Pattern： ^ 匹配符， 代表匹配所有
 - Definition： ha-mode=all 为匹配类型， 分为3种模式： all （表示所有的queue）

ha-mode： 指明镜像队列的模式， 可选下面的其中一个

 all： 表示在集群中所有的节点上进行镜像同步（一般都用这个参数）

 exactly： 表示在指定个数的节点上进行镜像同步， 节点的个数由ha-params指定

 nodes： 表示在指定的节点上进行镜像同步， 节点名称通过ha-params指定

ha-sync-mode： 镜像消息同步方式 automatic (自动) , manually (手动)

RabbitMQ™ RabbitMQ 3.8.9 Erlang 23.2.1

Refreshed 2021-02-15 17:20:20 Refresh every 5 seconds

Virtual host /dev Cluster rabbit@rabbit_host1 User admin Log out

Overview Connections Channels Exchanges Queues Admin

Policies

User policies

Add / update a policy

Virtual host: /dev Name: xdclass_policy Pattern: ^

Apply to: Exchanges and queues Priority:

Definition: ha-mode = all String ha-sync-mode = automatic String String

Queues [All types] Max length | Max length bytes | Overflow behaviour Dead letter exchange | Dead letter routing key HA mode | HA params | HA sync mode HA mirror promotion on shutdown | HA mirror promotion on failure Message TTL | Auto expire | Lazy mode | Master Locator

Queues [Classic] Max in memory length | Max in memory bytes | Delivery limit HA mode | HA params | HA sync mode HA mirror promotion on shutdown | HA mirror promotion on failure Message TTL | Auto expire | Lazy mode | Master Locator

Queues [Quorum] Max in memory length | Max in memory bytes | Delivery limit HA mode | HA params | HA sync mode HA mirror promotion on shutdown | HA mirror promotion on failure Message TTL | Auto expire | Lazy mode | Master Locator

Exchanges Alternate exchange Federation upstream set | Federation upstream

Add / update policy

Operator policies

Add / update an operator policy

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

- 配置好后，+2的意思是有三个节点，一个节点本身和两个镜像节点，且可以看到策略名称 xdclass_mirror

RabbitMQ™ RabbitMQ 3.8.9 Erlang 23.2.1

Refreshed 2021-01-17 23:15:08 Refresh every 5 seconds

Virtual host / Cluster rabbit@rabbit_host1 User guest Log out

Overview Connections Channels Exchanges Queues Admin

Queues

All queues (1)

Pagination

Page 1 of 1 - Filter: Regex ? Displaying 1 item , page size up to: 100

Overview							Messages			Message rates		
Virtual host	Name	Node	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
/	order_queue	rabbit@rabbit_host3	+2	classic	D xdclass_mirror	idle	0	0	0	0.00/s	0.00/s	

Add a new queue

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

- 集群重启顺序
 - 集群重启的顺序是固定的，并且是相反的
 - 启动顺序：磁盘节点 => 内存节点
 - 关闭顺序：内存节点 => 磁盘节点
 - 最后关闭必须是磁盘节点，否则容易造成集群启动失败、数据丢失等异常情况

第7集 SpringBoot2.x项目整合RabbitMQ高可用mirror镜像集群

简介：使用SpringBoot项目整合RabbitMQ高可用集群

- SpringBoot AMQP单机配置

```
spring:  
  rabbitmq:  
    host: 10.211.55.13  
    port: 5672  
    virtual-host: /  
    password: guest
```

```
username: guest
# 投递到交换机
publisher-confirm-type: correlated
# 交换机到队列
publisher-returns: true
##指定消息在没有被队列接收时是否强行退回还是直接丢弃,true是退回
template:
    mandatory: true

#开启手动确认消息，如果消息重新入对则会一直重试，可以配置重试次数
listener:
    simple:
        acknowledge-mode: manual
```

- 高可用镜像集群配置

```
#消息队列
spring:
  rabbitmq:
    addresses:
      10.211.55.13:5672,10.211.55.13:5673,10.211.55.13:5674
    virtual-host: /dev
    password: xdclass.net168
    username: admin
    #开启消息二次确认,生产者到broker的交换机
    publisher-confirm-type: correlated
```

```
#开启消息二次确认，交换机到队列的可靠性投递
publisher-returns: true
#为true，则交换机处理消息到路由失败，则会返回给生产者
template:
    mandatory: true

#消息手工确认ACK
listener:
    simple:
        acknowledge-mode: manual
```

- 高可用集群测试
 - 关闭消费者监听
 - 生产者发送一个消息
 - 停止节点一和节点二，web管控台访问不了
 - 启动消费者监听，可以消费到消息



愿景："让编程不再难学，让技术与生活更加有趣"

第十五章 高级面试题+原理篇幅-玩转大厂里面的RabbitMQ核心技能面试

- 持续更新中，2021年3月初完结



小滴课堂，愿景：让编程不在难学，让技术与生活更加有趣

相信我们，这个是可以让你学习更加轻松的平台，里面的课程绝对会让你技术不断提升

欢迎加小D讲师的微信： **xdclass-lw**

我们官方网站：<https://xdclass.net>

千人IT技术交流QQ群： **718617859**

重点来啦：加讲师微信 免费赠送你干货文档大集合，包含前端，后端，测试，大数据，运维主流技术文档（持续更新）

<https://mp.weixin.qq.com/s/qYnjcDYGFDQorWmSfE7lpQ>



零基础到Java高级开发工程师 架构全套学习路线

适合人群

- ✓ 零基础自学, 到高级Java后端工程师就业路线;
- ✓ 传统软件工程师 转型 互联网项目技术栈;
- ✓ 学习时间约1~2个月, 涵盖几十套核心课程, 基础+多个项目实战
- ✓ 一线城市平均薪酬15~25k

阶段一

○ Java工程师零基础就业班学习路线

- 1 新版Javase零基础到高级实战
- 2 全新HTML+CSS零基础入门到进阶网页制作教程
- 3 全新Javascript零基础多实战例子教程
- 4 Linux/Centos7零基础入门到高级实战视频教程
- 5 全新Mysql数据库零基础入门到实战精讲
- 6 新版Javaweb+jdbc+maven零基础到开发论坛项目实战

7 【面试必备】多线程并发编程+JUC+JDK源码教程

8 新版SSM框架-SpringBoot2.3+Spring5+Mybatis3.x
零基础到开发小滴课堂移动端系统综合实战

9 正版Redis4.x零基础整合springboot视频教程

阶段二

○中级后端工程师路线

1 IDEA零基础入门到进阶(整合阿里巴巴编码规范)

2 新版Maven3.5+Nexus私服搭建全套核心技术

3 SpringBoot 2.x微信支付在线教育网站项目实战

4 前端-后端必备-Nginx零基础到分布式高并发专题

5 Jenkins持续集成 Git Gitlab Sonar视频教程

6 微服务SpringCloud+Docker入门到高级实战

7 JMeter接口压力测试打造高性能服务

8 Redis高并发高可用集群百万级秒杀实战

9 权限框架Shiro+SpringBoot2.x零基础到高级实战

10 全新JDK8~JDK13全套新特性教程

阶段三

○ 高级后端工程师/技术经理路线

1 Docker实战视频教程入门到高级

2 互联网架构之JAVA虚拟机JVM零基础到高级实战

3 Linux/shell脚本编程视频教程

4 玩转搜索框架ElasticSearch7.x实战

5 Zookeeper+Dubbo微服务教程分布式视频教程

6 RocketMQ4.X教程消息队列

**7 SpringBoot2微服务Dubbo优惠券项目实战
Java架构全套视频教程**

**8 互联网架构之Netty4.X入门到进阶打造百万连接
服务器**

9 上进逆袭全栈工程师学习手册和项目实战手册

- X D C L A S S . N E T -

扫码咨询客服小姐姐 >>

开启属于你的Java高级开发之路

