**activeMq**

# 简介

## ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现

## 什么是MQ

## 什么是JMS

## MQ的基本组成

### 生产者

### 消息

* 消息头

• JMSDestination(目的地)

• JMSDeliveryMode(传递模式是否持久化)

• 默认不开启

• JMSExpiration(有效期)

• 默认永久不失效

• JMSPriority(优先级)

• 0-9十个级别

• 0-4是普通消息

• 5-9是加急消息

• 默认级别为4

• JMSMessageID(消息ID)

• 唯一识别每个消息的标识由MQ产生

• 幂等性，避免重复消费

* 消息体

• 封装具体的消息数据

• 五种消息体格式

• textMessage

• 普通字符串消息。包含一个String

• MapMessage

• 一个Map类型的消息，key为String类型，而值为java的基本类型

• BytesMessage

• 二级制数组消息，包含一个byte()

• StreamMessage

• java数据流消息，用标准流操作来顺序的填充和读取

• ObjectMessage

• 对象消息，包含一个可序列化java对象

• 发送和接受的消息体类型必须一致对应

* 消息属性

• 如果需要除消息头字段以外的值，那么可以使用消息属性

• 识别/去重/重点标注等操作非常有用的方法

• 是什么

• 属性名/属性值绑定的kv键值对，拓展信息识别作用，类型有String和基本类型等

### MQ服务器

### 消费者

# 解决问题

## 核心问题:系统之间直接调用实际工程落地和存在的问题

### 系统之间接口耦合比较严重

* 系统业务调用链过长，且相互依赖严重

### 面对大流量并发时，容易被冲垮

* 每个接口的吞吐量有限

### 等待同步存在性能问题

* RPC接口基本是同步调用，整体的服务性能遵循"木桶理论"，即整体系统的耗时取决于链路中最慢的那个接口。那么如果部分接口耗时过长，会影响整个服务性能

• 额外知识点:RPC框架的理解

### ...

## 解决策略

### 系统解耦

* 当新的模块接入的时候，改动代码最小

### 能够削峰

* 设置流量缓冲池，按照后天服务器自身吞吐量进行消费

### 能够异步

* 强弱依赖梳理能将非关键调用链路的操作一步花并提升整体系统的吞吐能力

## 工作经验

### 作业服务--强制提交作业流程拥有同步性能问题需要用activeMQ进行异步操作

### ...

# 模式

## 生产/消费(Queue)模式

## 发布/订阅(Topic)模式

### 注意点

* 订阅要在发布前启动

## 对比

### 1.queue是点对点模式，一条消息对应一个消费者，topic是一对多模式，一条消息可能有一个或多个消费者

### 2.queue模式消息再发送后消费者可以在之后的任意时间消费，topic模式如果没有订阅者消息就是废消息，会被丢弃。

### 3.queue模式生产者与消费者之间没有时间相关性，topic模式下生产者和消费者之间有一定的时间相关性，消费者只能接收到订阅之后的生产者发送的消息。

# 定义

## 利用高效可靠的消息传递机制与平台无关的数据交流，并基于数据通信来进行分布式系统的集成

## 通过消息传递和消息排队模型在分布式环境下应用解耦，弹性伸缩，冗余存储，流量削峰，异步通信，数据同步等功能

## 大致过程就是:发送者把消息发送给消息服务器，消息服务器将消息存放在若干队列QUEUE/主题TOPIC中，在合适的时候，消息服务器会将消息转发给接受者。在这个过程中是异步的，且能完成一对多通信

# 实现安装

## 实现目的

### 解决业务问题

### 最主要的问题:实现高可用，高性能，可伸缩，易用等企业级面向消息的应用系统

### 异步消息的消费和处理

### 控制消息的消费顺序

### 可以和spring/springBoot做个整合简化编码

### 配置集群和容错的MQ集群

### ...

## 实现

### 基础信息

* URL

• tcp链接

* name

### 基础流程

* 创建链接工厂
* 打开链接
* 创建会话
* 创建目的地(创建队列或者模式)

• 队列

• 1：1消费

• 持久化指定消费

• 如果有两个消费者则类似负载均衡平均分配消费队列

• 自支持消息落地

• 主题

• 1:N消费

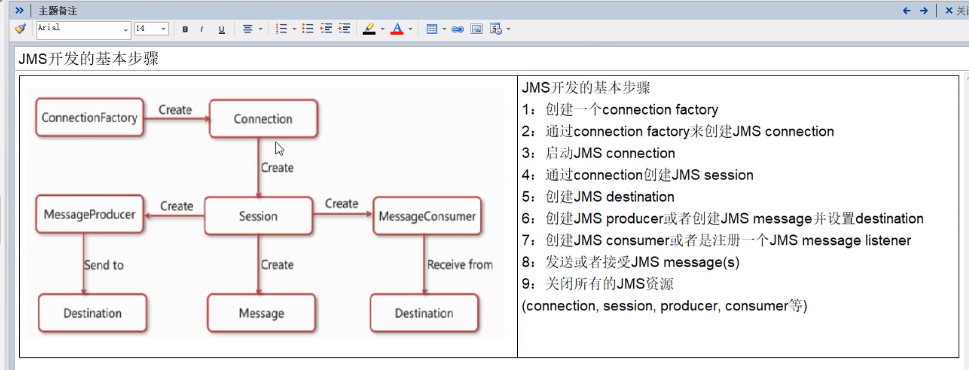
• 时间上相关性，只会推送订阅之后的消息

• 无状态不落地，消息废除机制

• 持久化操作需要额外配置

* 创建生产者/消费者
* 执行生产或者消费逻辑
* 关闭资源

### 流程图解



### 生产者

* send方法

### 消费者

* receive接收

• 有参:执行时间

• 无参:永久等待

* messageListener监听器

• 注意:给消费流程，不然可能未消费就直接关闭资源了

## broker

### 是什么

* 相当于一个activeMq服务器实例
* 嵌入的形式嵌入到java代码中，以便随时启动，节省资源

### 不同配置文件模拟不同的实例

### 嵌入式broker

## 传输协议

### 是什么

* 各种协议有各自擅长该协议的中间件，工作中一般不会使用activemq去实现这些协议。如： mqtt是物联网专用协议，采用的中间件一般是mosquito。ws是websocket的协议，是和前端对接常用的，一般在java代码中内嵌一个基站（中间件）。stomp好像是邮箱使用的协议的，各大邮箱公司都有基站（中间件）

•



### 种类

* TCP
* NIO

• <transportConnector name="nio" uri="nio://0.0.0.0:61616"/>

* AMQP
* STOMP
* MQTT
* WS
* AUTO+NIO

• <transportConnector name="auto+nio" uri="auto+nio://0.0.0.0:61608?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600&amp;org.apache.activemq.transport.nio.SelectorManager.corePoolSize=20&amp;org.apache.activemq.transport.nio.Se1ectorManager.maximumPoo1Size=50"/>

* java

### 性能优化

* 原本conf包下面都是的协议配置都是BIO同步阻塞的协议更改为NIO同步非阻塞就能提升一定性能
* 减少配置使用AUTO+NIO能够适配所有协议且都是nio

# 缺点

## 配置性能低，需要优化配置，同时配置文件复杂，本身不提供管理工具

## 示例代码少，demo缺

## 文档齐全但是可读性较差没有规范的整理和统一

# 特性

## 消息可靠性

### 持久化

* 定义:将信息以另一种方式保存在JMS服务器上，但是如果使用主题持久化，则持久化保存的信息一定是消费者已经订阅过但是没有消费的消息
* 模式持久化

• 参数配置

• 持久化

• messageProducer.setDeliveryMode(DeliveryMode.PERSISTENT);

• 服务器宕机消息不丢失

• 非持久

• messageProducer.setDeliveryMode(DeliveryMode.NOT\_PERSISTENT);

• 服务器宕机消息不存在

• 队列持久化

• 默认持久化，因为一对一消费所以默认以数据可靠性为重要参考依据

• 主题持久化

• 代码

• 先启动订阅再启动生产

• 持久的发布主题生产者

• 持久的订阅主题消费者

• 链接开始前一定要先设置持久化，这样开启一个持久化的发布订阅消息

• 离线持久化

• 控制台

• 类似微信公众号订阅发布

* 消息持久化

• 持久化的配置，在${activemq\_home}/conf/activemq.xml配置文件中，标签 broker内部的 persistenceAdapter 中配置持久化方案

• 保证宕机数据不丢失

• 持久化策略

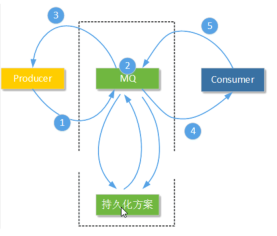
• JDBC(重点)

• 消息基于JDBC存储机制

• 能够真实进入mysql数据库

• 实现原理

•



• 实现步骤

• 添加mysql数据库的驱动包到lib文件

• jdbc持久化配置

•



• <!-- <persistenceAdapter> <kahaDB directory="${activemq.data}/kahadb"/> </persistenceAdapter>--><persistenceAdapter> <jdbcPersistenceAdapter dataSource="#mysql-ds" createTableOnStartup="true"/> </persistenceAdapter>

• 准备数据库连接池配置

• 建库和创表

• activeMq数据库

• 数据表

• activemq\_msgs

• activeMq\_acks

• activeMq\_lock

• 字段等自己拓展

• 总结

• ACK存主题

• msgs存消息

• 坑

• 导包和配置的时候注意名称，尤其是下划线

• activeMq自带的建表机制记得第一次用完之后要改写

• 配合 JDBC MESSAGE store with ActiveMQ Journal

• 高速缓存

• 提前消费

• 在同步DB之前存储缓存

• 缓存消费

• 然后把没消费完的同步数据库

• 前提

• 消费者消费速度快

• kahadb

• 基于日志文件，activeMq5.4开始流行

• 类似redis AOF

• 内部文件

• db-1.log

• 32M

• db.data

• BTree索引

• db.redo

• 恢复和容灾机制

• db.free

• lock

• 文件锁，获取当前的broker

• 持久化原理

• 日志

• 悲观锁

• 容灾

• 归档删除

• AMQ

• 基于文件存储方式，现在暂时不用

• LevelDB

• 也是基于本地日志存储，基本跟kahadb一致，但是使用的不是Btree而是自身支持的索引

• 好处

• 数据更新

• 并发读

• 链路跟踪

• 但是还在持续更新中，暂时不会推荐使用

### 事务

* 核心:事务偏生产者，签收偏消费者
* producer提交时的事务

• true

• 先执行send在执行commit，消息才被真正提交到队列中

• 消息需要批量发送，需要缓冲区处理

• false

• 只需要执行send，就进入队列中

• 关闭事务，那第二个签收参数的设置需要有效

* consumer签收时的事务

• true

• 消费过后必须要使用commit，不然可能会造成重复消费

• false

• 只会被消费一次

### 签收

* 非事务

• 自动签收(默认)

• AUTO\_ACKNOWLEDGE

• 手动签收

• CLIENT\_ACKNOWLEDGE

• 客户端调用acknowledge方法手动签收

• message.acknowledge();

• 如果不用方法无法消费信息

• 允许重复消息

• DUPS\_OK\_ACKNOWLEDGE

* 事务

• 生产事务开启，只有commit后才能将全部消息变为已消费

• 消息生产者

• 代码

• 并未太大改动

• 控制台

• 消费消费者

• 代码

• 事务大于签收(commit>acknowledge)

• 控制台

* 签收和事务关系

• 在事务会话中，当一个事务被成功提交则消息被自动签收，如果事务回滚，则消息会被再次传送

• 非事务性会话中，消息何时被确认取决于创建会话时候的应答模式(acknowledge)

### connection.createSession(事务,签收)

## 测试相关

### 内嵌JMS provider

# 高可用+集群

## master-slave

### 文件共享实现

### 数据库实现

### levelDB实现

## 局域zookeeper和levelDB搭建的activeMq集群，避免单点故障

## zookeeper+replicated-leveldb-store的主从集群

## 配置流程

### 环境版本

### 关闭防火墙

### zk集群配置启动

### 集群部署规划表

### 创建3台集群目录

### 修改控制台端口

### hostname名字映射

### 集群配置

### 修改各节点的消息端口

### zk集群节点状态说明

## 故障迁移测试

# 额外功能拓展

## 高可用性

### 持久化

### 事务签收

### 集群

## 异步投递

### 对于是一个慢的消费者，同步发送消息可能会导致堵塞，慢消费者适用

### 特点

* 没有使用事务且发送的是持久化消息，每一次发送没有绑定事务切发送持久化消息
* 允许少量数据丢失情况
* 大幅度提升生产者性能

### 缺点

* 降低activeMq服务器性能
* 不能确保消息准确发送

## 延迟投递和定时投递

### 配置文件修改

* schedulerSupport="true"

### 代码修改

* // 延迟的时间

• textMessage.setLongProperty(ScheduledMessage.AMQ\_SCHEDULED\_DELAY, delay);

* // 重复投递的时间间隔

• textMessage.setLongProperty(ScheduledMessage.AMQ\_SCHEDULED\_PERIOD, period);

* // 重复投递的次数

• textMessage.setIntProperty(ScheduledMessage.AMQ\_SCHEDULED\_REPEAT, repeat);

### 具体考量业务场景是否需要

## 消息重试

## 死信队列

## 幂等性

## 分发策略

# spring相关配置

## 后续有时间拓展

# springBoot配置

## springboot配置文件配置

### # ACTIVEMQ (ActiveMQProperties)spring.activemq.broker-url= # URL of the ActiveMQ broker. Auto-generated by default. spring.activemq.close-timeout=15s # Time to wait before considering a close complete. spring.activemq.in-memory=true # Whether the default broker URL should be in memory. Ignored if an explicit broker has been specified. spring.activemq.non-blocking-redelivery=false # Whether to stop message delivery before re-delivering messages from a rolled back transaction. This implies that message order is not preserved when this is enabled. spring.activemq.password= # Login password of the broker. spring.activemq.send-timeout=0ms # Time to wait on message sends for a response. Set it to 0 to wait forever. spring.activemq.user= # Login user of the broker. spring.activemq.packages.trust-all= # Whether to trust all packages. spring.activemq.packages.trusted= # Comma-separated list of specific packages to trust (when not trusting all packages). spring.activemq.pool.block-if-full=true # Whether to block when a connection is requested and the pool is full. Set it to false to throw a "JMSException" instead. spring.activemq.pool.block-if-full-timeout=-1ms # Blocking period before throwing an exception if the pool is still full. spring.activemq.pool.enabled=false # Whether a JmsPoolConnectionFactory should be created, instead of a regular ConnectionFactory. spring.activemq.pool.idle-timeout=30s # Connection idle timeout. spring.activemq.pool.max-connections=1 # Maximum number of pooled connections. spring.activemq.pool.max-sessions-per-connection=500 # Maximum number of pooled sessions per connection in the pool. spring.activemq.pool.time-between-expiration-check=-1ms # Time to sleep between runs of the idle connection eviction thread. When negative, no idle connection eviction thread runs. spring.activemq.pool.use-anonymous-producers=true # Whether to use only one anonymous "MessageProducer" instance. Set it to false to create one "MessageProducer" every time one is required.

## spring源码配置

### DefaultJmsListenerContainerFactoryConfigurer

* 解析器

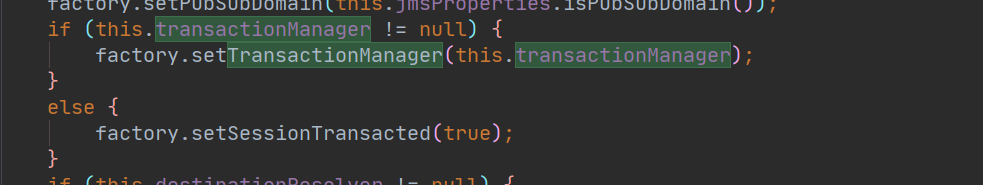
• DestinationResolver

• resolveDestinationName

• 用于解析JMS目的地的策略接口

* 事务

•



• 默认开启事务

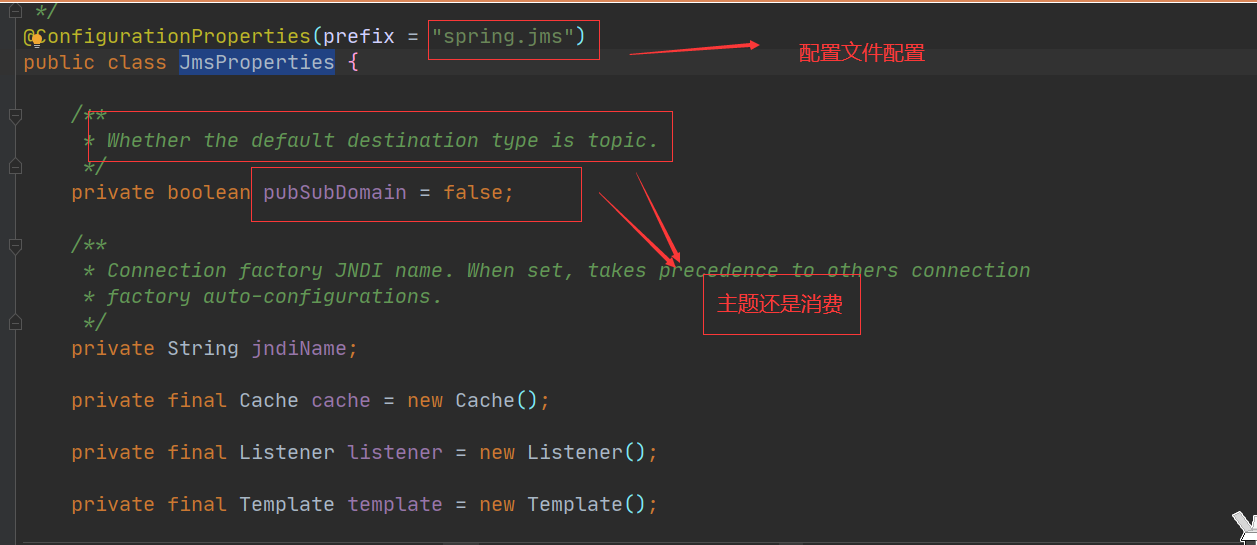
* JMS

• JmsProperties

• 基础配置

• 基础配置

•



* 消息转换器

• messageConverter

• toMessage

• fromMessage

• 指定Java对象和JMS消息之间的转换器的策略接口。

### @JmsListener注解

* 参数相关

• destination

• 此侦听器的目标名称，通过容器范围目的地

• containerFactory

• 用于创建负责服务此端点的消息侦听器容器。

• subscription

• 持久订阅的名称(如果有)

• selector

• JMS消息选择器表达式(如果有)。有关选择器表达式的详细定义，请参见JMS规范。

• concurrency

• 并发限制，如果有需要查询官网配置

* 拓展内容

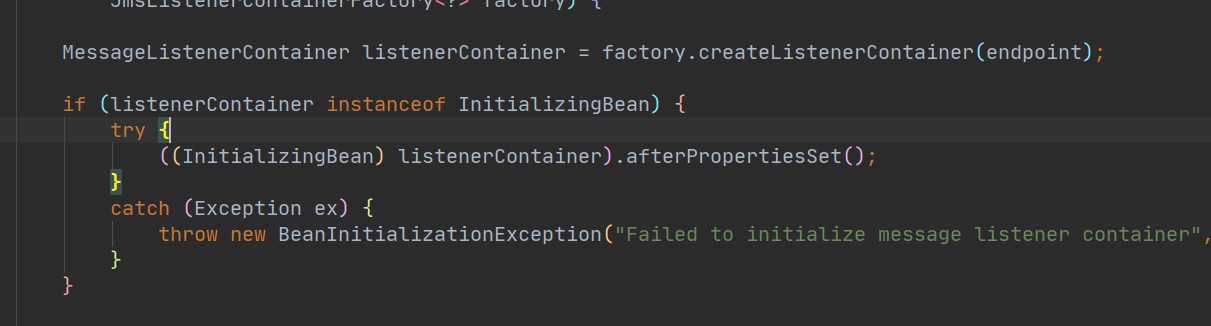
• 监听机制

• JmsListenerEndpointRegistry

• JmsListenerEndpointRegistry

• 什么时候开启监听

•

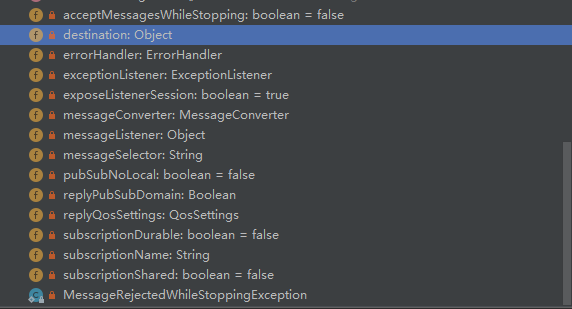


• 核心(基础监听器)

• AbstractMessageListenerContainer

• 默认jms监听机制不会失败，异常是不会传递，但是如果有需要异常的机制，那么需要errorHandler完成

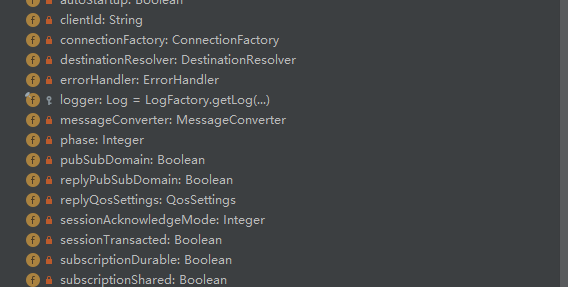
•



• AbstractJmsListenerContainerFactory

• createListenerContainer

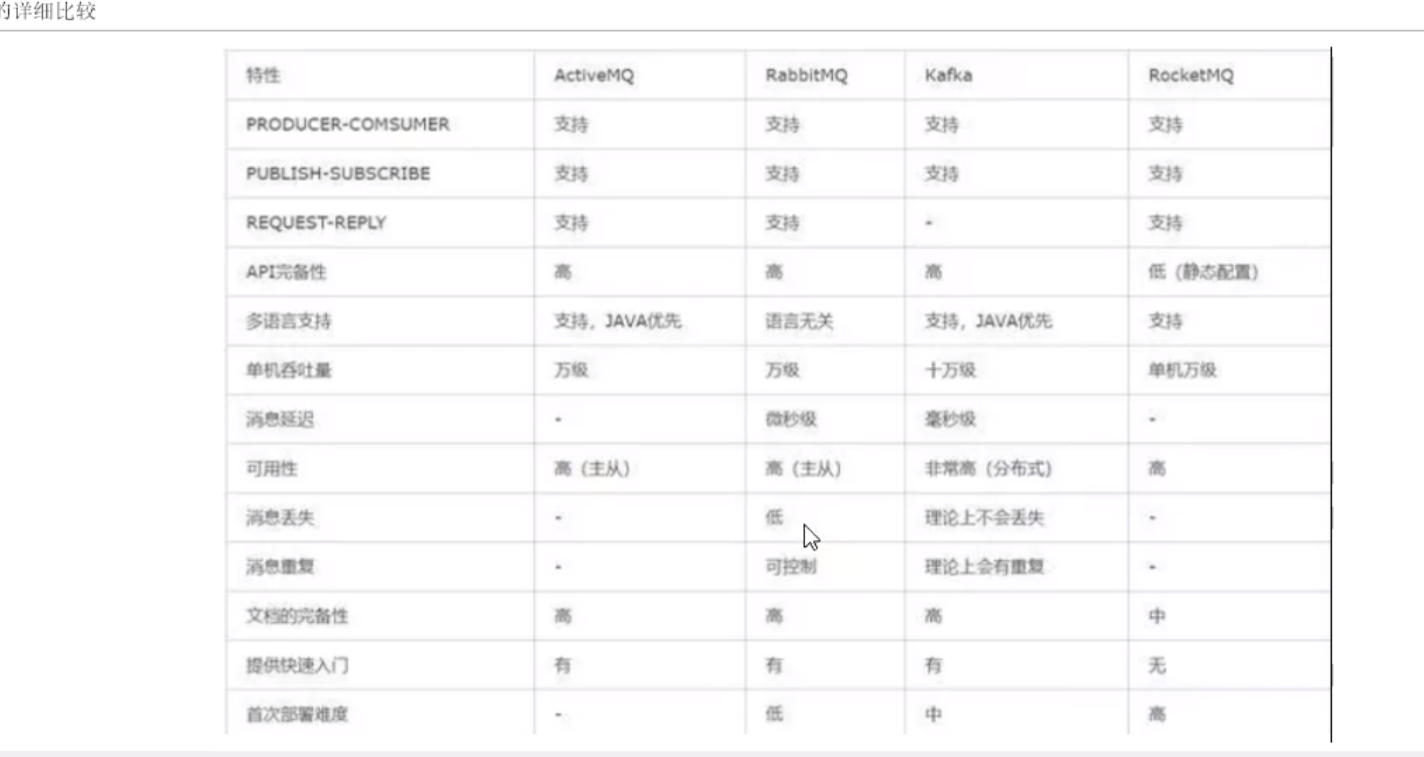
•



### ActiveMqConfigBean

* //自己的配置写法支持springBoot@Configurationpublic class ActiveMqConfigBean { @Value("${spring.activemq.user}") private String usrName; @Value("${spring.activemq.password}") private String password; @Value("${spring.activemq.broker-url}") private String brokerUrl; @Bean("connectionFactory") public CachingConnectionFactory connectionFactory() { CachingConnectionFactory connectionFactory = new CachingConnectionFactory(); ActiveMQConnectionFactory activeMQConnectionFactory = new ActiveMQConnectionFactory(usrName, password, brokerUrl); //配置异步投递--但是要写回调函数确认接收成功 activeMQConnectionFactory.setUseAsyncSend(true); //配置重试机制是否需要 RedeliveryPolicy redeliveryPolicy = new RedeliveryPolicy(); redeliveryPolicy.setMaximumRedeliveries(3); activeMQConnectionFactory.setRedeliveryPolicy(redeliveryPolicy); //消费者预取 ActiveMQPrefetchPolicy prefetchPolicy = new ActiveMQPrefetchPolicy(); prefetchPolicy.setQueuePrefetch(1000); prefetchPolicy.setMaximumPendingMessageLimit(100000); activeMQConnectionFactory.setPrefetchPolicy(prefetchPolicy); //批量确认 activeMQConnectionFactory.setOptimizeAcknowledge(true); connectionFactory.setTargetConnectionFactory(activeMQConnectionFactory); //会话并发需求--不用原生连接池，无法共享，每次链接都要创建 connectionFactory.setSessionCacheSize(100); return connectionFactory; } @Bean("jacksonJmsMessageConverter") public MessageConverter jacksonJmsMessageConverter() { MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter(); converter.setTargetType(MessageType.TEXT); converter.setTypeIdPropertyName("\_type"); return converter; } @Bean("jmsTemplate") JmsTemplate jmsTemplate(@Qualifier("jacksonJmsMessageConverter") MessageConverter jacksonJmsMessageConverter, @Qualifier("connectionFactory") CachingConnectionFactory connectionFactory) { JmsTemplate jmsTemplate = new JmsTemplate(connectionFactory); jmsTemplate.setMessageConverter(jacksonJmsMessageConverter); //设置优先级，0最低，9最高 jmsTemplate.setPriority(9); return jmsTemplate; } @Bean("jmsMessagingTemplate") JmsMessagingTemplate jmsMessagingTemplate(@Qualifier("jacksonJmsMessageConverter") MessageConverter jacksonJmsMessageConverter, @Qualifier("jmsTemplate") JmsTemplate jmsTemplate) { JmsMessagingTemplate messagingTemplate = new JmsMessagingTemplate(jmsTemplate); messagingTemplate.setJmsMessageConverter(jacksonJmsMessageConverter); return messagingTemplate; } @Bean("jmsListenerContainerQueue") public JmsListenerContainerFactory<?> jmsListenerContainerQueue(@Qualifier("connectionFactory") CachingConnectionFactory connectionFactory) { DefaultJmsListenerContainerFactory bean = new DefaultJmsListenerContainerFactory(); bean.setConnectionFactory(connectionFactory); //设置消息转换器 bean.setMessageConverter(jacksonJmsMessageConverter()); return bean; } @Bean("jmsListenerContainerTopic") public JmsListenerContainerFactory<?> jmsListenerContainerTopic(@Qualifier("connectionFactory") CachingConnectionFactory connectionFactory) { DefaultJmsListenerContainerFactory bean = new DefaultJmsListenerContainerFactory(); //设置为发布订阅方式, 默认情况下使用的生产消费者方式 bean.setPubSubDomain(true); bean.setConnectionFactory(connectionFactory); //设置消息转换器 bean.setMessageConverter(jacksonJmsMessageConverter()); return bean; }}

# 



## 比较MQ