

第8章 RocketMQ消息中间件

广东财经大学信息学院

罗 东 俊 博士

ZSUJONE@126.COM

(内部资料，请勿外传)



目的和要求

- 了解为什么要使用消息中间件。
- 熟悉RocketMQ消息中间件的基本概念和工作原理。
- 掌握Spring Boot与RocketMQ的整合实现与应用。



主要内容

- 8.1 消息服务概述
- 8.2 RocketMQ基础
- 8.3 RocketMQ整合案例



8.1 消息服务概述

■ 8.1.1 认识消息服务

■ 8.1.2 常见消息中间件



8.1.1认识消息服务

- 1. 消息服务作用
- 2. 消息服务存在的问题



1. 消息服务作用

■ 使用消息服务中间件处理业务能够提升系统的**异步通信**和**扩展解耦**能力，从而可以实现一个**高性能、高可用、高扩展**的系统。

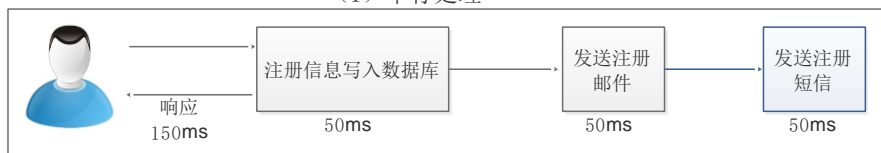
■ 常见的应用场景：

- ✧①异步处理
- ✧②应用解耦
- ✧③流量削锋
- ✧④消息通讯
- ✧⑤分布式事务管理等。



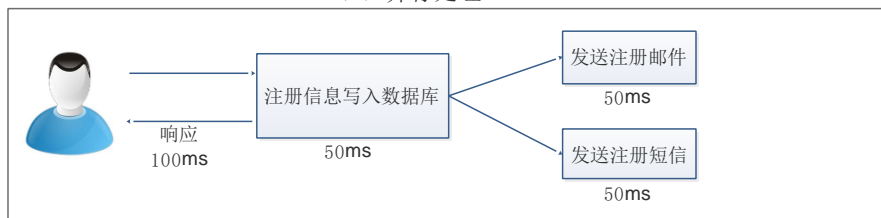
①异步处理

(1) 串行处理



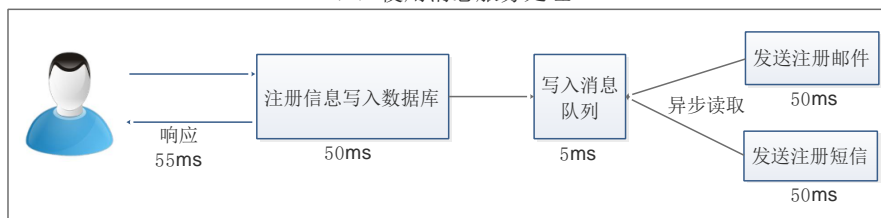
耗时，用户体验不好

(2) 并行处理



较为耗时的业务处理，不够完善

(3) 使用消息服务处理



快速



②应用解耦

(1) 传统处理方式



库存系统出现异常，订单服务会失败导致订单丢失

(2) 使用消息服务



订单服务的下订单消息会快速写入消息队列，库存服务会监听并读取到订单，从而修改库存

高效可靠



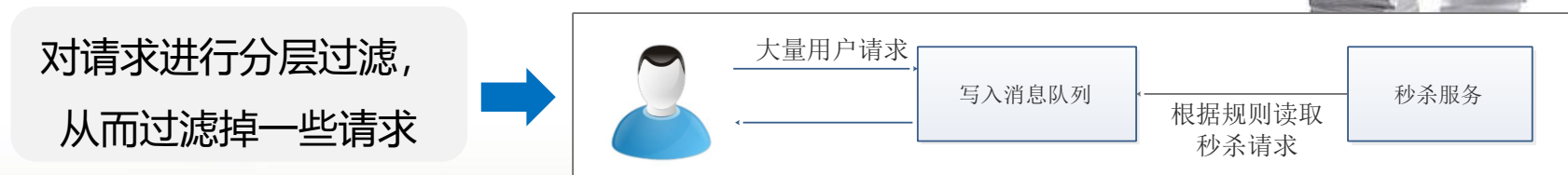
③流量削峰

■ 服务器处理资源能力有限，出现峰值时很容易造成服务器宕机、用户无法访问的情况。

■ 解决办法：

✧ 服务器接收用户请求后，首先写入消息队列，如果消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面。

✧ 之后秒杀业务根据规则对消息队列中的请求再做后续处理。



④消息通讯

■消息队列一般都内置了高效的通信机制，因此也可应用在纯的消息通讯中。

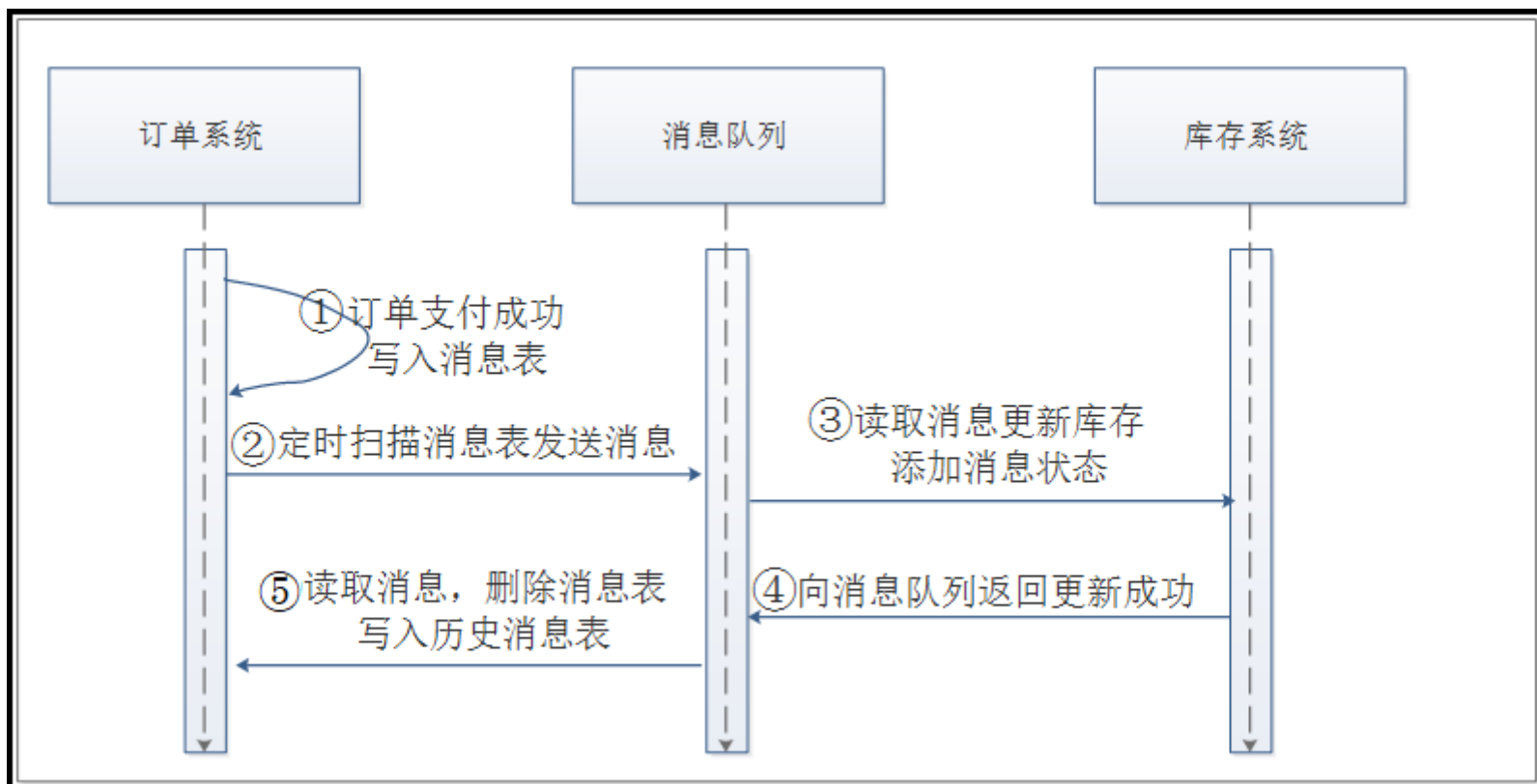
■例如：

✧点对点通讯：客户端**A**和客户端**B**使用同一队列进行消息通讯，只有一个发送者和一个接收者。

✧聊天室通讯：客户端**A**，客户端**B**，客户端**N**订阅同一主题，进行消息发布和接收。



⑤分布式事务管理



分布式事务管理流程

- 订单支付成功后，写入消息表
- 定时扫描消息表消息写入到消息队列中
- 库存系统会立即读取到消息队列中的消息进行库存更新，同时添加消息处理状态
- 库存系统向消息队列中写入库存处理结果
- 订单系统会立即读取到消息队列中的库存处理状态。直接删除消息表数据，并写入到历史消息表



2.消息服务存在的问题

■ ①系统可用性降低

- ✧ 系统引入的外部依赖越多，系统稳定性越差。一旦MQ宕机，就会对业务造成影响。
- ✧ 如何保证MQ的高可用？

■ ②系统复杂度提高

- ✧ MQ的加入大大增加了系统的复杂度，以前系统间是同步的远程调用，现在是通过MQ进行异步调用。
- ✧ 如何保证消息没有被重复消费？怎么处理消息丢失情况？如何保证消息传递的顺序性？

■ ③一致性问题

- ✧ A系统处理完业务，通过MQ给B、C、D三个系统发消息数据，如果B系统、C系统处理成功，D系统处理失败。
- ✧ 如何保证消息数据处理的一致性？



8.1.2 常见消息中间件

特性	ActiveMQ	RabbitMQ	RocketMQ	Kafka
单机吞吐量	万级，比 RocketMQ、Kafka 低一个数量级	同 ActiveMQ	10 万级，支撑高吞吐	10 万级，高吞吐，一般配合大数据类的系统来进行实时数据计算、日志采集等场景
topic 数量对吞吐量的影响			topic 可以达到几百/几千的级别，吞吐量会有较小幅度的下降，这是 RocketMQ 的一大优势，在同等机器下，可以支撑大量的 topic	topic 从几十到几百个时候，吞吐量会大幅度下降，在同等机器下，Kafka 尽量保证 topic 数量不要过多，如果要支撑大规模的 topic，需要增加更多的机器资源
时效性	ms 级	微秒级，这是 RabbitMQ 的一大特点，延迟最低	ms 级	延迟在 ms 级以内
可用性	高，基于主从架构实现高可用	同 ActiveMQ	非常高，分布式架构	非常高，分布式，一个数据多个副本，少数机器宕机，不会丢失数据，不会导致不可用
消息可靠性	有较低的概率丢失数据	基本不丢	经过参数优化配置，可以做到 0 丢失	同 RocketMQ
功能支持	MQ 领域的功能极其完备	基于 erlang 开发，并发能力很强，性能极好，延时很低	MQ 功能较为完善，还是分布式的，扩展性好	功能较为简单，主要支持简单的 MQ 功能，在大数据领域的实时计算以及日志采集被大规模使用

常见消息中间件

		ActiveMQ	RocketMQ	RabbitMQ	Kafka
定位	设计定位	可靠消息传输	非日志的可靠消息传输	可靠消息传输	实时数据处理以及日志处理
基础对比	成熟度	成熟	成熟	成熟	日志领域成熟
	所属社区/公司	Apache	Alibaba开发 现已加入 Apache	Apache	Apache
	社区活跃度	高	中	高	高
	API 完备性	高	高	高	高
	开发语言	Java	Java	ErLang	Scala
	支持协议	OpenWire、STOMP、REST XMPP、AMQP	自己定义的一套(社区提供JMS--不成熟)	AMQP	一套自行设计的基于TCP的二进制协议
	持久化方式	内存/文件/数据库	磁盘文件	内存/文件	磁盘文件
	客户端支持语言	Java、C、C++、Python、PHP、Perl、.net 等	Java C++(不成熟)	Java、C、C++、Python、PHP、Perl、.net 等	C、C++、Python、Go、PHP 等

可用性/性能比较	部署方式	单机/集群	单机/集群	单机/集群	单机/集群
	集群管理	独立	nameserver	独立	zookeeper
	选主方式	基于zookeeper+leavelDB的master-slave方式	支持多master模式,多master多slave模式,异步复制模式	Master 提供服务, slave 提供备份	支持多副本机制,leader宕机,flower自动顶上重新选举leader
	可用性	高	非常高	高	非常高
	消息写入性能	较好	很好	较好	非常好
功能对比	单机队列数	较好	单机最高5万	依赖内存	单机超过64个队列或分区会出现飙升
	事务消息	支持	支持	不支持	不支持
	消息过滤	不支持	支持	不支持	不支持
	消息查询	不支持	支持	不支持	不支持
	消息失败重试	支持	支持	支持	不支持
	消息重新消费	支持	支持	不支持	支持
	消费方式	Consumer pull	Consumer pull	Broker push	Consumer pull
	批量发送	支持	支持	不支持	支持
	消息清理	指定文件保存时间过期删除	指定文件保存时间过期删除	可用内存少于40%出发gc	指定文件保存时间过期删除

8.2 RocketMQ基础

- 8.2.1 RocketMQ概述
- 8.2.2 RocketMQ单节点安装
- 8.2.3 RocketMQ工作原理
- 8.2.4 RocketMQ应用模式



8.2.1 RocketMQ概述

- 1.RocketMQ简介
- 2.RocketMQ基本概念
- 3.RocketMQ系统架构



1.RocketMQ简介

- **RocketMQ**是阿里巴巴采用**Java语言**开发的开源分布式消息中间件，现在是**Apache**的一个顶级项目。
- 在阿里内部，**RocketMQ**承接了例如“双11”等高并发场景的消息流转，能够处理万亿级别的消息。
- 跟其它中间件相比，**RocketMQ**的特点是：
 - ✧ 纯**JAVA**实现；
 - ✧ 集群和**HA**（高可用性）实现相对简单；
 - ✧ 在发生宕机和其它故障时消息丢失率更低。



2.RocketMQ基本概念

- 1) 消息 (Message)
- 2) 主题 (Topic)
- 3) 标签 (Tag)
- 4) 队列 (Queue)



1) 消息 (Message)

- **消息 (Message)**：消息系统所传输信息的物理载体，生产和消费数据的最小单位，每条消息必须属于一个主题。
- **RocketMQ**中的每个消息拥有唯一的**MessageID**，且可携带由用户指定的业务相关的唯一标识**Key**，系统提供通过**MessageID**和**Key**查询消息的功能。
 - ✧ 系统中的**Index Files**提供了一种可通过业务标识**Key**或时间区间来查询消息的方法。



2) 主题 (Topic)

- **主题 (Topic)**：表示一类消息的集合，每个主题包含若干条消息，每条消息只能属于一个主题，是**RocketMQ**进行消息订阅的基本单位。

- ✧ **topic:message=1:n**

- ✧ **message:topic=1:1**

- 一个生产者可以同时发送多种**Topic**的消息；而一个消费者只对某种特定的**Topic**感兴趣，即只可以订阅和消费一种**Topic**的消息。

- ✧ **producer:topic=1:n**

- ✧ **consumer:topic=1:1**



3) 标签 (Tag)

- 为消息设置的标签，用于同一主题下区分不同类型的消息。
 - ✧ **Topic**是消息的一级分类，**Tag**是消息的二级分类。
- 消费者可以根据**Tag**实现对**不同子主题的不同消费逻辑**，实现更好的扩展性。



4) 队列 (Queue)

■ 在RocketMQ中存在两种队列:

✧消息队列 (**Message Queue**): 是真正存放消息单元的物理实体, 即系统中的**CommitLog Files**

- 采用混合型存储结构, 不分**Topic**按消息接收顺序依次将消息写入日志文件

✧消费队列 (**Consume Queue**): 是存放指向消息队列中消息单元的索引的物理实体, 可据此来查找待消费的消息。

- 按**Topic**分类将消息单元索引写入队列文件



Consume Queue

■ 一个Topic中可以包含多个Consume Queues，每个Consume Queue中存放的就是该Topic消息单元的索引。

✧ 一个Consume Queue也被称为一个分区（Partition）

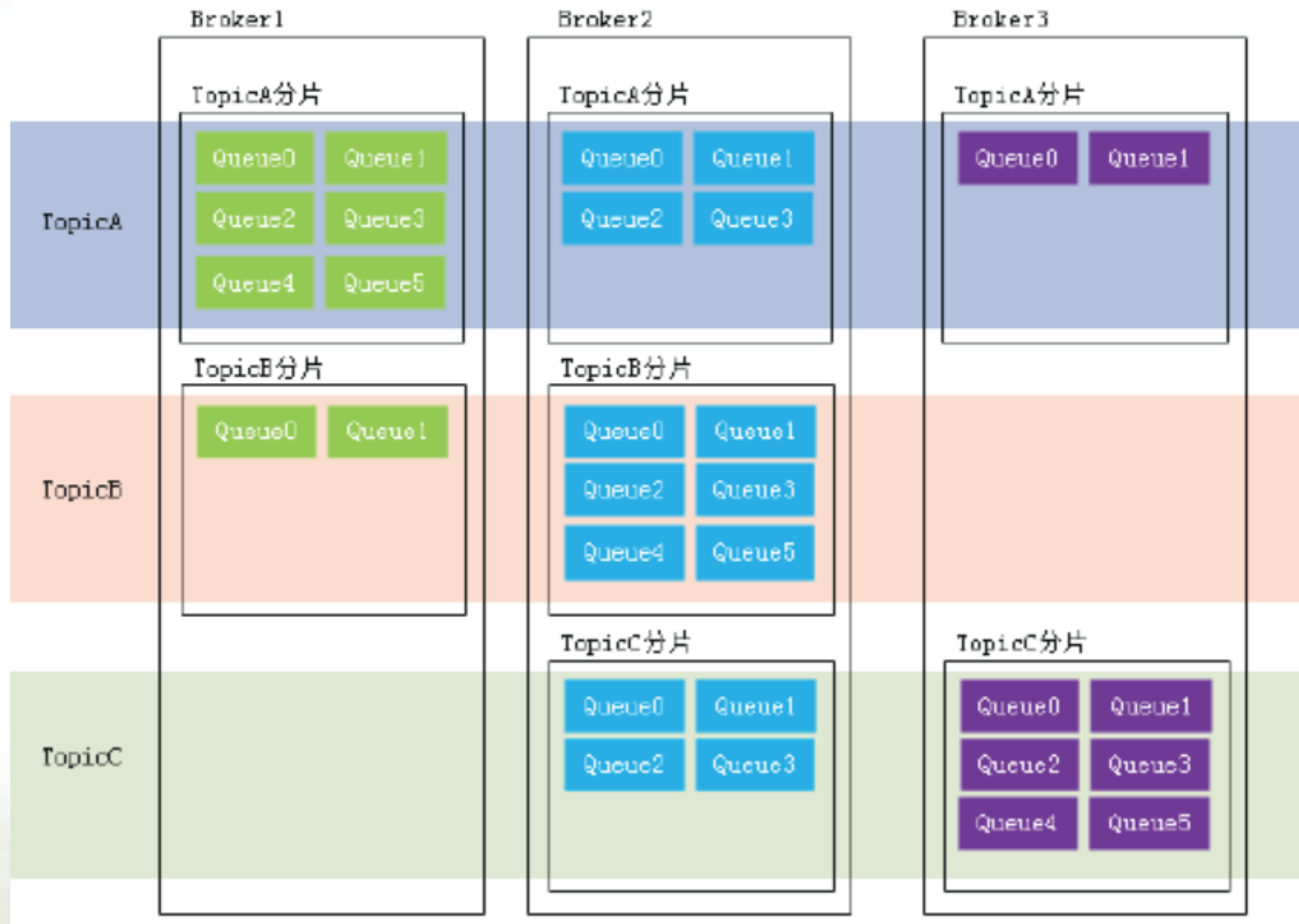
- 一个分区中的消息只能被一个消费者组中的一个消费者消费。

✧ 存放相应Topic的Broker也被称为分片（Sharding）

- 每个分片中会创建出相应数量的分区，即Consume Queues，每个Consume Queue的大小都是相同的。



Consume Queue

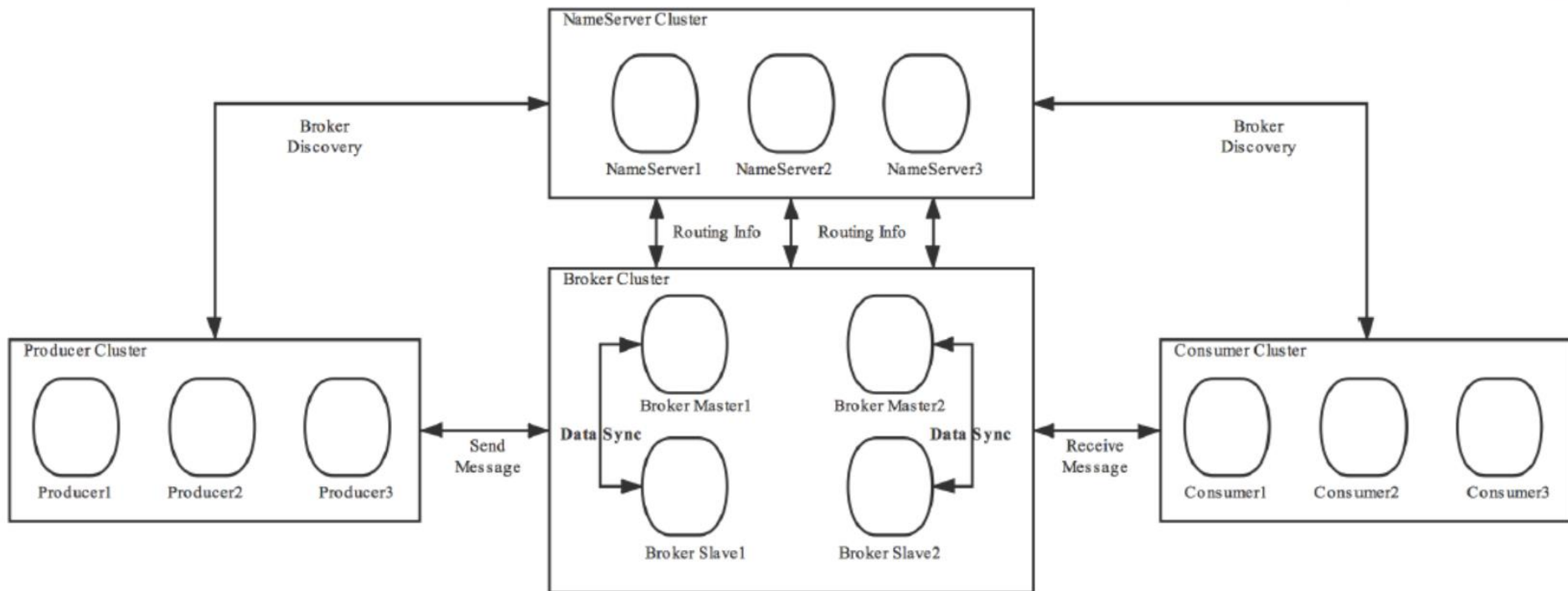


3.RocketMQ系统架构

- 1) Producer
- 2) Consumer
- 3) NameServer
- 4) Broker



系统架构



1) Producer

■ 消息生产者，负责生产和发送消息。

✧ **RocketMQ**提供多种消息发送方式，如同步发送、异步发送、顺序发送、单向发送。

✧ 同步和异步方式均需**Broker**返回确认信息，单向发送则不需要。

■ 消息生产者都是以生产者组（**Producer Group**）的形式出现。

✧ 生产者组是同一类生产者的集合，这类**Producer**发送相同**Topic**类型的消息。

✧ 一个生产者组可以同时发送多个主题的消息。



2) Consumer

■ 消息消费者，负责消费消息。

✧ 一个消息消费者会从**Broker**服务器中获取到消息，并对消息进行相关业务处理。

■ 消息消费者都是以消费者组（**Consumer Group**）的形式出现的。

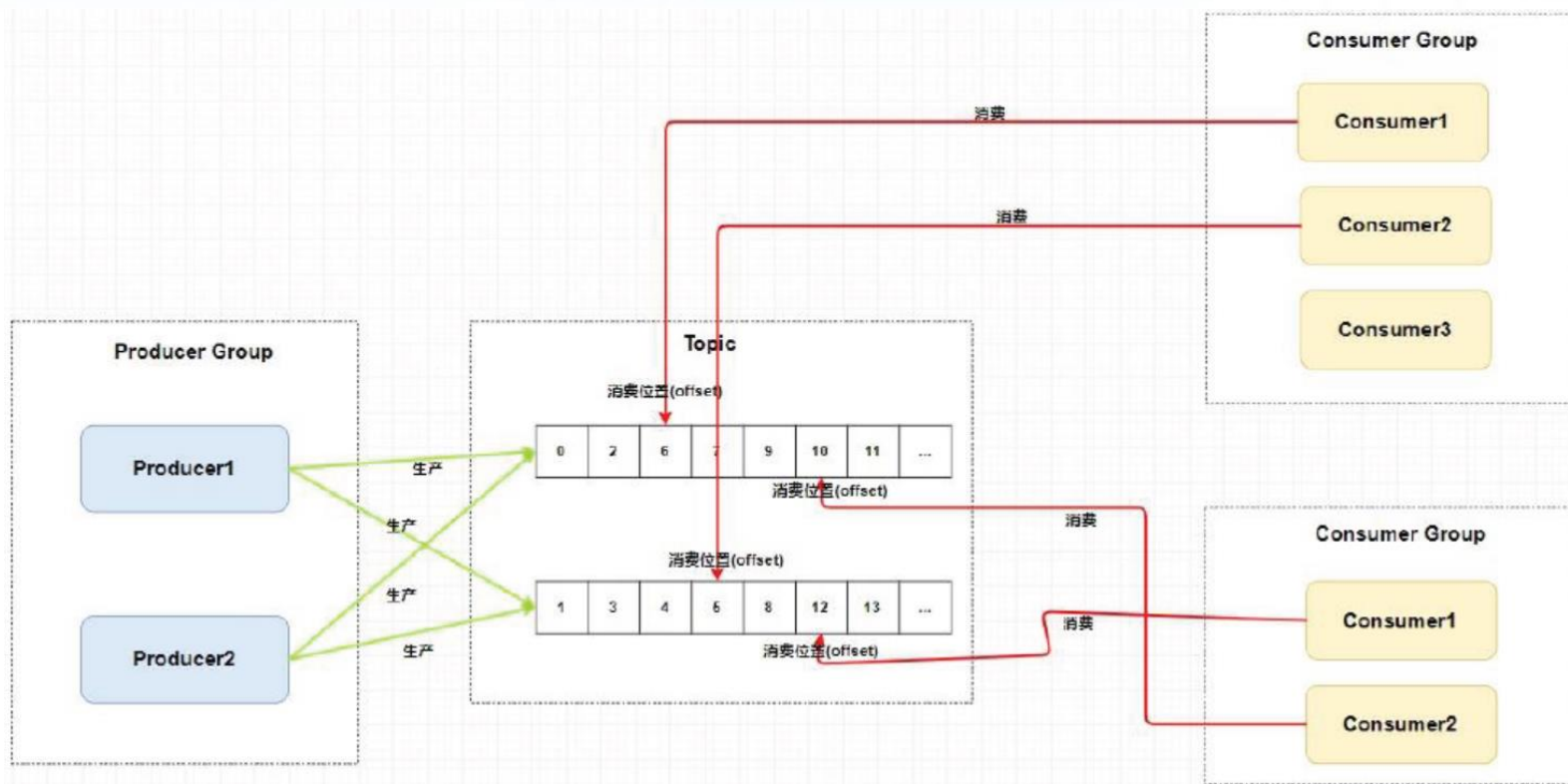
✧ 消费者组是同一类消费者的集合，这类**Consumer**消费的是同一个且只能一个**Topic**类型的消息，但一个**Topic**类型的消息可以被多个消费者组同时消费。

✧ 消费者组使得在消息消费方面，实现**负载均衡**和**容错**的目标变得非常容易。

- 负载均衡：将一个**Topic**中的不同的**Queue**平均分配给同一个**Consumer Group**的不同的**Consumer**（注意：并不是将消息负载均衡）
- 容错：一个**Consumer**挂了，该**Consumer Group**中的其它**Consumer**可以接着消费原**Consumer**消费的**Queue**



消费者组消费消息例子



超出Queue数量的Consumer将不能消费消息；且在集群消费模式下，每条同Topic消息只会被Consumer Group中的某个Consumer消费。

3) NameServer

■ **NameServer**是一个**Broker**和**Topic**路由注册中心，支持**Broker**的动态注册与发现。

✧ **NameServer**内部维护着一个**Topic**路由表和一个**Broker**列表；并提供心跳检测机制，检查**Broker**是否还存活。

■ **NameServer**通常也是以集群的方式部署，但各节点间相互不进行信息通讯。

✧ 在**Broker**节点启动时，轮询**NameServer**列表，与每个**NameServer**节点建立长连接，定时注册**Topic**路由信息到所有**NameServer**。



Topic路由表

Topic	QueueData	
	BrokerName	QueueId
Topic1	BrokerName1	0
		1
		2
		3
	BrokerName2	0
		1
		2
		3
Topic2	BrokerName1	0
		1
		2
		3
	BrokerName2	0
		1
		2
		3
...



Broker列表

BrokerName	BrokerData		
	BrokerName	BrokerId	BrokerIP
BrokerName1	BrokerName1	0(Master)	192.168.1.120
		1(Slave1)	192.168.1.120
		2(Slave2)	192.168.1.121
BrokerName2	BrokerName2	0(Master)	192.168.1.121
		1(Slave1)	192.168.1.121
		2(Slave2)	192.168.1.120
...

具有相同BrokerName的Broker构成一个Broker主备集群；不同Broker主备集群又可构成一个大Broker集群。



路由剔除

■ **Broker**节点为了证明自己是活着的，会每**30秒**向**NameServer**发送一次心跳包。

✧心跳包中包含**BrokerId**、**Broker地址(IP+Port)**、**Broker名称**、**Broker所属集群名称**、存储的所有**Topic**信息等等。

✧**NameServer**在接收到心跳包后，会更新心跳时间戳，记录这个**Broker**的最新存活时间。

■ **NameServer**中有一个定时任务，每隔**10秒**就会扫描一次**Broker**列表，查看每一个**Broker**的最新心跳时间戳距离当前时间是否超过**120秒**，如果超过，则会判定**Broker**失效，然后将其从**Broker**列表中剔除。

手动路由剔除

■ 对于RocketMQ日常运维工作，例如Broker升级，需要停掉Broker的工作，OP（运维工程师）需要怎么做？

✧ OP需要在RocketMQ控制台将Broker的读写权限禁掉。

- 一旦client(Consumer或Producer)向broker发送请求，都会收到broker的NO_PERMISSION响应，然后client会进行对其它Broker的重试。

✧ 当OP观察到这个Broker没有流量后，再关闭它，实现Broker从NameServer的移除。



路由发现

■ RocketMQ的路由发现采用的是Pull模型。

- ✧ 当Topic路由信息出现变化时，NameServer不会主动推送（Push模型）给客户端，而是客户端定时拉取（Pull模型）主题最新的路由。
- ✧ 默认客户端每30秒会拉取一次最新的路由。

■ 三种模型比较：

✧ Push模型

- 实时性较好，需要维护一个长连接

✧ Pull模型

- 不需要维护一个长连接，但实时性较差

✧ Long Polling模型：长轮询模型。

- 是对Push与Pull模型的整合，充分利用了这两种模型的优势，屏蔽了它们的劣势。



客户端选择NameServer策略

■ 首先采用**随机**策略，失败后采用**轮询**策略。

✧ 客户端首先会生产一个随机数，然后再与**NameServer**集群中的节点数量取模，结果即为所要连接的节点索引，然后进行连接。

✧ 如果连接失败，则会采用**round-robin**策略，逐个尝试着去连接其它节点。



4) Broker

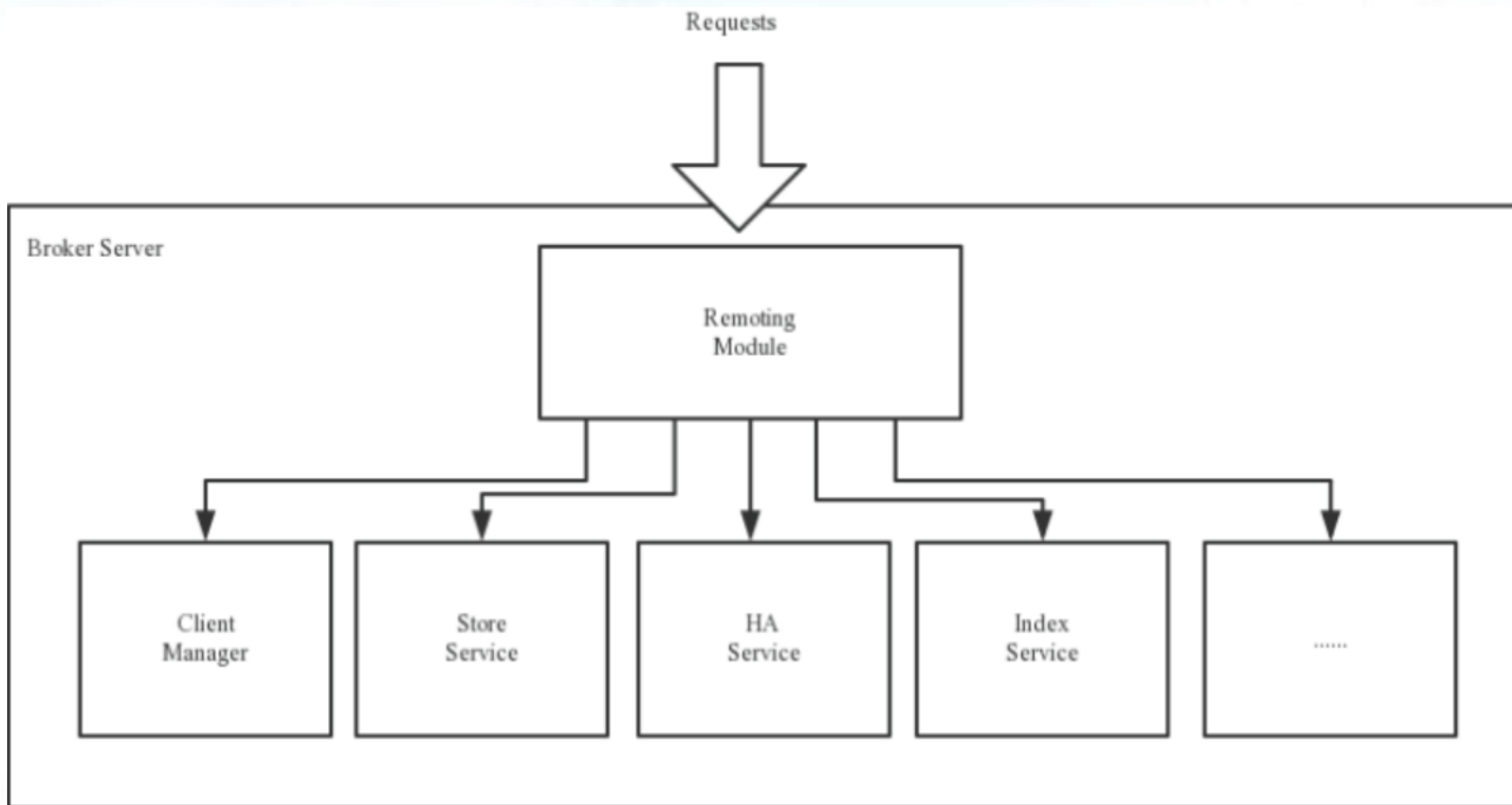
■ **Broker**充当着消息中转角色，负责消息的存储和转发，同时负责消息的查询和服务高可用性保证。

✧接收并存储从生产者发送来的消息，同时为消费者的拉取请求作准备。

■ **Broker**同时也存储着消息相关的元数据，包括消费者组消费进度偏移**offset**、主题、队列等。



Broker Server的功能模块



Broker Server的功能模块

■ **Remoting Module**是整个Broker的实体，负责处理来自clients端请求。

■ 这个Broker实体由以下模块构成：

✧ **Client Manager**：客户端管理器。

- 负责接收、解析客户端(Producer/Consumer)请求，管理客户端。
- 例如，维护Consumer的Topic订阅信息

✧ **Store Service**：存储服务。

- 提供方便简单API接口，处理消息存储到物理硬盘和消息查询功能。

✧ **HA Service**：高可用服务。

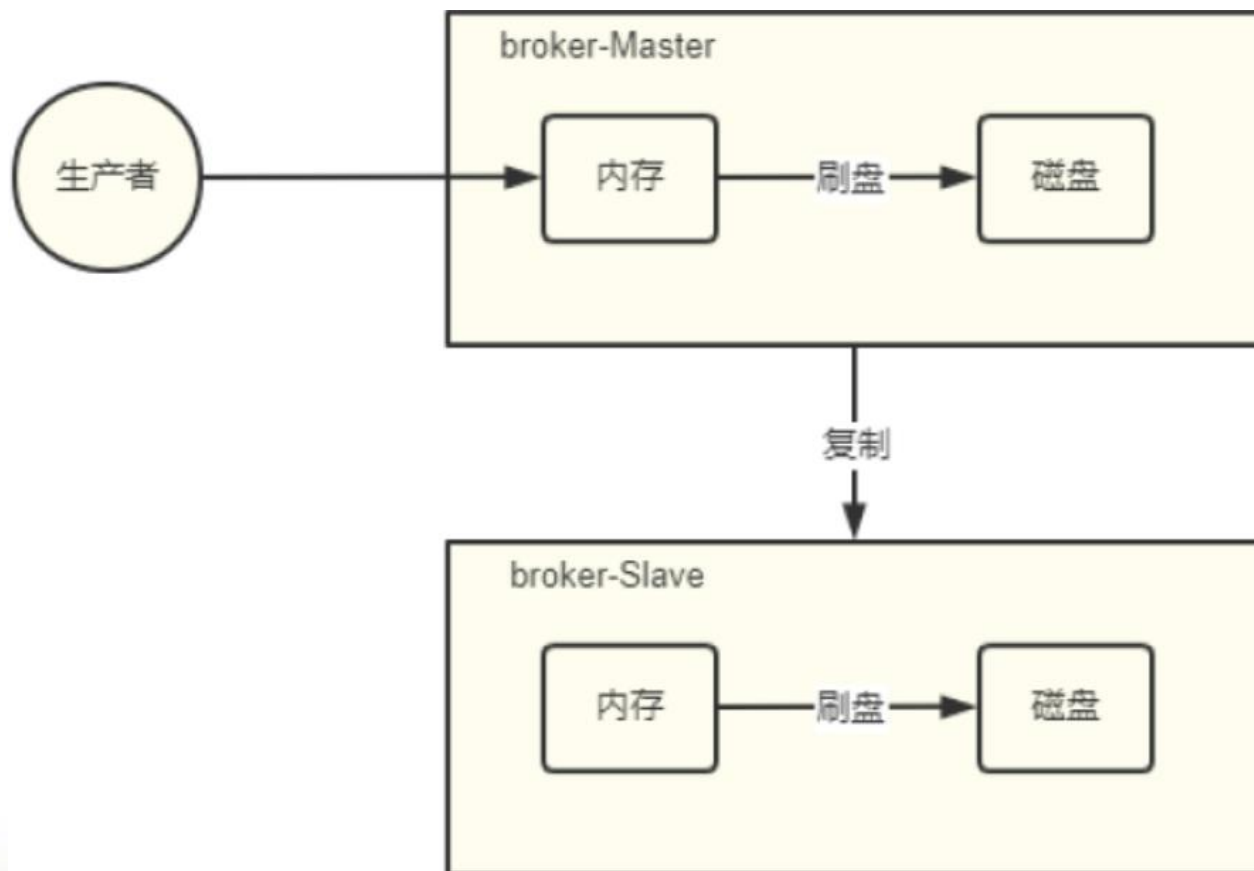
- 提供Master Broker 和 Slave Broker之间的数据同步功能。

✧ **Index Service**：索引服务。

- 根据特定的Message key，对投递到Broker的消息进行索引服务，同时也提供根据Message Key对消息进行快速查询的功能。



Broker中数据的复制与刷盘策略



复制策略

■ 复制策略是**Broker**的**Master**与**Slave**间的数据同步方式，分为：

✧ 同步复制：消息写入**master**后，**master**会等待**slave**同步数据成功后才向**producer**返回成功**ACK**

✧ 异步复制：消息写入**master**后，**master**立即向**producer**返回成功**ACK**，无需等待**slave**同步数据成功



刷盘策略

■ 刷盘策略指的是**broker**中消息的落盘方式，即消息发送到**broker**内存（一般是**PageCache**）后消息持久化到磁盘的方式，分为：

- ✧ 同步刷盘：当消息持久化到**broker**的磁盘后才返回成功**ACK**。
- ✧ 异步刷盘：当消息写入到**broker**的内存后立即返回成功**ACK**，无需等待消息持久化到磁盘。当写入**PageCache**到达一定量时会自动进行落盘。



8.2.2 RocketMQ单节点安装

- 1. 下载RocketMQ
- 2. 安装RocketMQ中间件
- 3. 安装RocketMQ控制台



1. 下载RocketMQ

■ 下载网址：

✧ <https://downloads.apache.org/rocketmq/>

■ 下载文件：

✧ **rocketmq-all-4.9.2-bin-release.zip**

✧ **rocketmq-dashboard-1.0.0-source-release.zip**



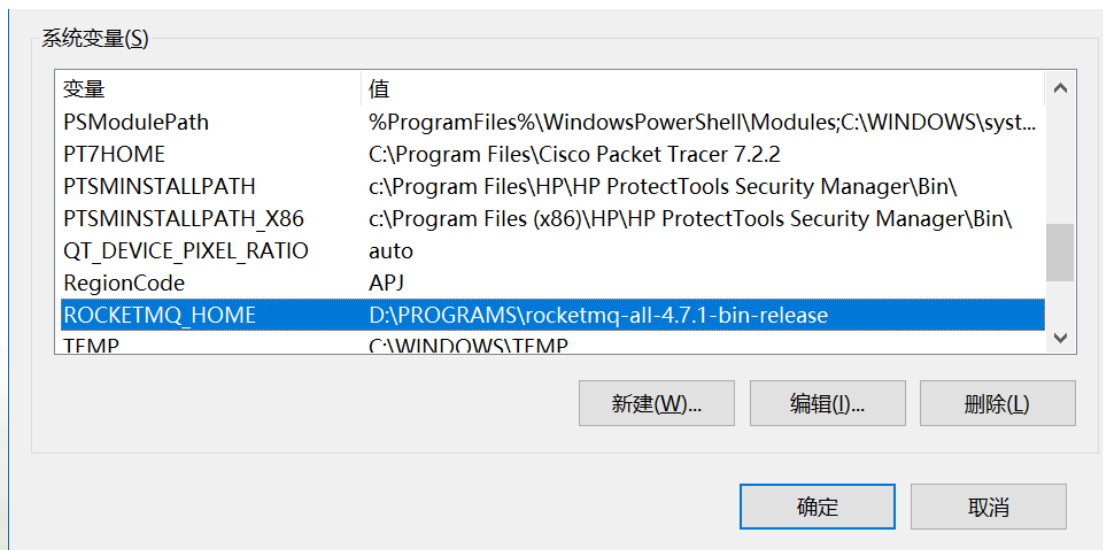
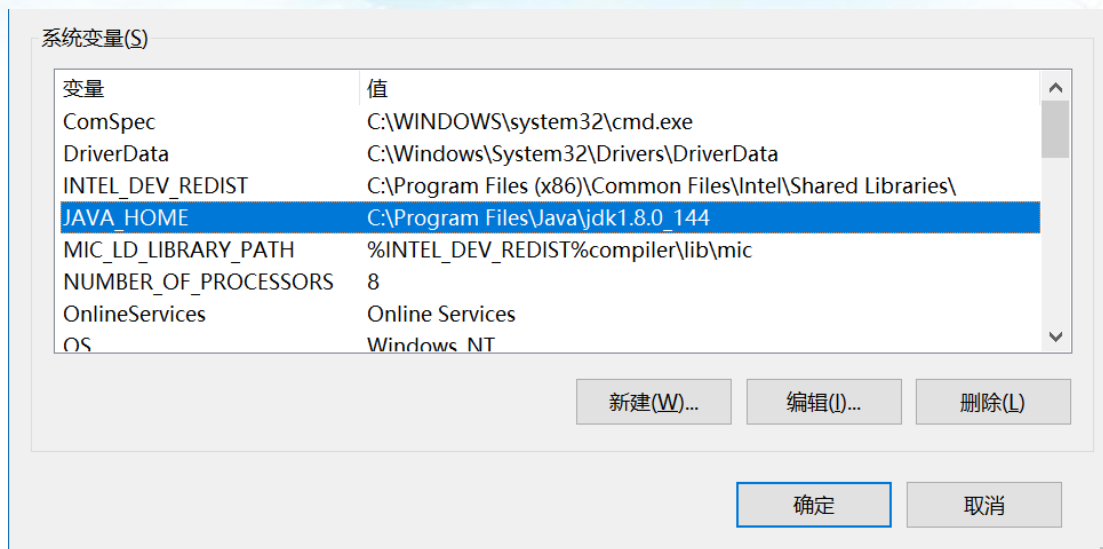
2. 安装RocketMQ中间件

■ 安装RocketMQ步骤:

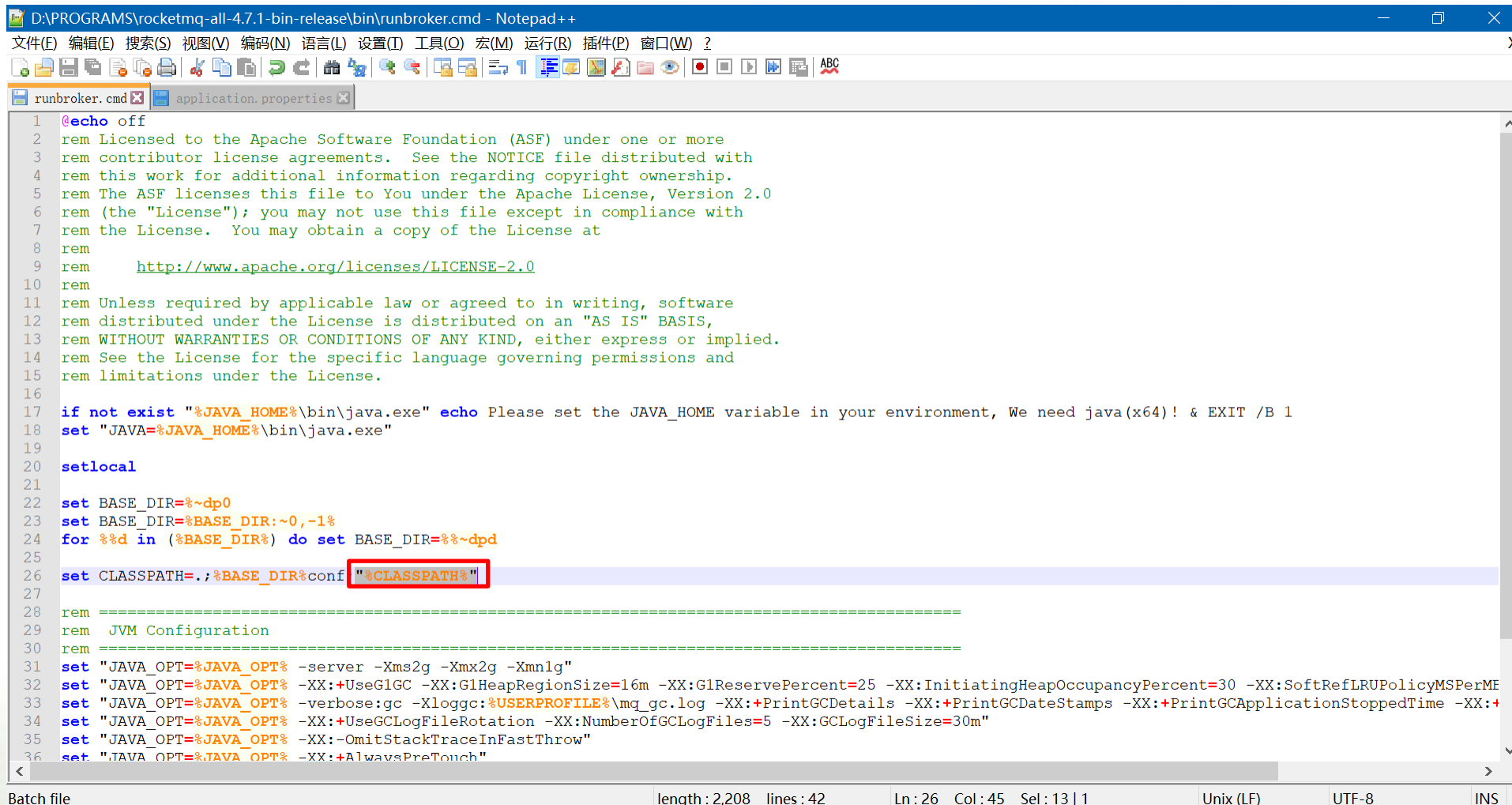
- ✧ ① 设置环境变量
- ✧ ② 编辑runbroker.cmd
- ✧ ③ 启动NameServer和Broker



①设置环境变量



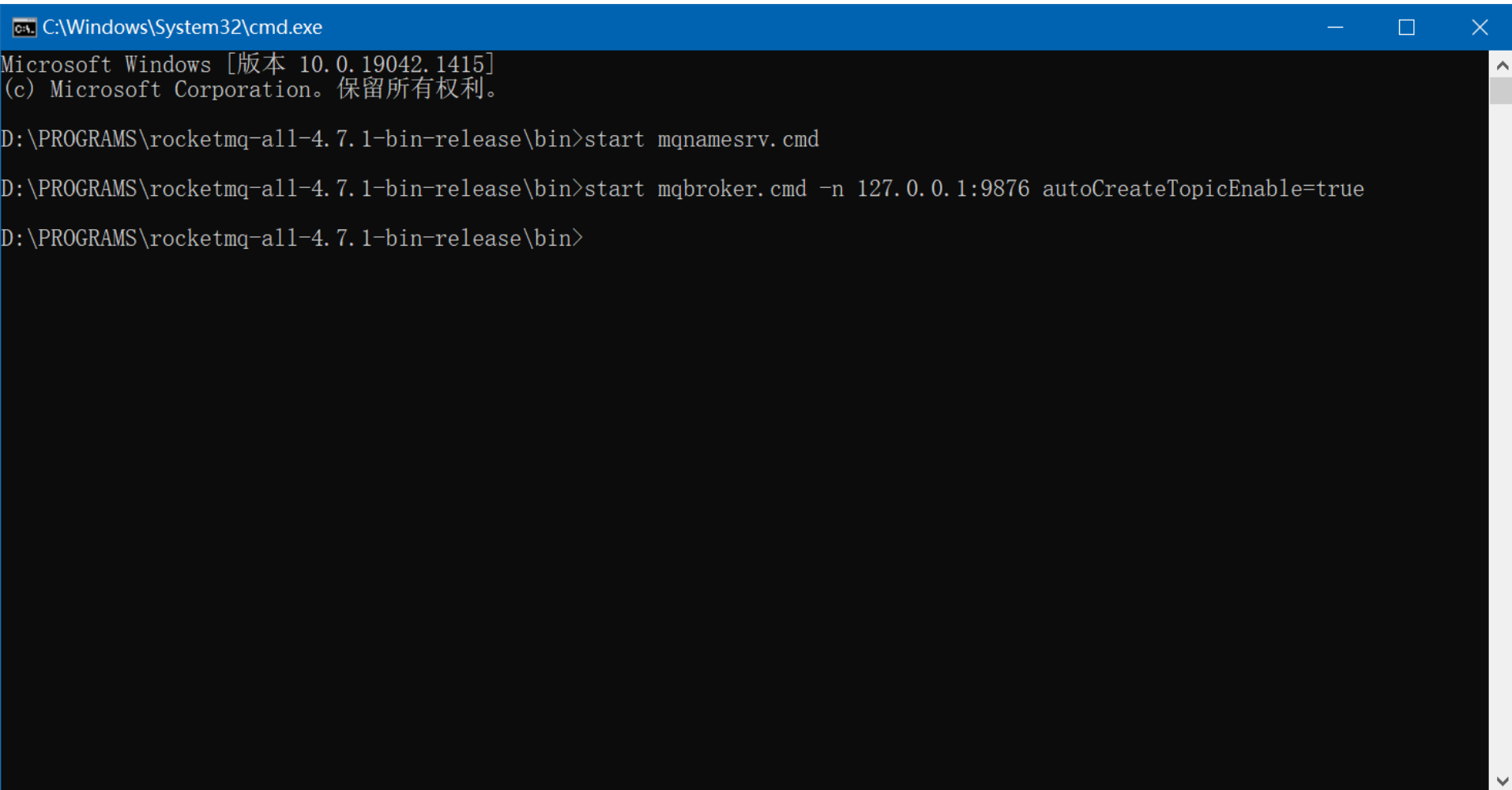
②编辑runbroker.cmd



```
1 @echo off
2 rem Licensed to the Apache Software Foundation (ASF) under one or more
3 rem contributor license agreements. See the NOTICE file distributed with
4 rem this work for additional information regarding copyright ownership.
5 rem The ASF licenses this file to You under the Apache License, Version 2.0
6 rem (the "License"); you may not use this file except in compliance with
7 rem the License. You may obtain a copy of the License at
8 rem
9 rem http://www.apache.org/licenses/LICENSE-2.0
10 rem
11 rem Unless required by applicable law or agreed to in writing, software
12 rem distributed under the License is distributed on an "AS IS" BASIS,
13 rem WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 rem See the License for the specific language governing permissions and
15 rem limitations under the License.
16
17 if not exist "%JAVA_HOME%\bin\java.exe" echo Please set the JAVA_HOME variable in your environment, We need java(x64)! & EXIT /B 1
18 set "JAVA=%JAVA_HOME%\bin\java.exe"
19
20 setlocal
21
22 set BASE_DIR=%~dp0
23 set BASE_DIR=%BASE_DIR:~0,-1%
24 for %%d in (%BASE_DIR%) do set BASE_DIR=%%~dpd
25
26 set CLASSPATH=.;%BASE_DIR%\conf "%CLASSPATH%"
27
28 rem =====
29 rem JVM Configuration
30 rem =====
31 set "JAVA_OPT=%JAVA_OPT% -server -Xms2g -Xmx2g -Xmn1g"
32 set "JAVA_OPT=%JAVA_OPT% -XX:+UseG1GC -XX:G1HeapRegionSize=16m -XX:G1ReservePercent=25 -XX:InitiatingHeapOccupancyPercent=30 -XX:SoftRefLRUPolicyMSPerMB
33 set "JAVA_OPT=%JAVA_OPT% -verbose:gc -Xloggc:%USERPROFILE%\mq_gc.log -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCApplicationStoppedTime -XX:+
34 set "JAVA_OPT=%JAVA_OPT% -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=30m"
35 set "JAVA_OPT=%JAVA_OPT% -XX:-OmitStackTraceInFastThrow"
36 set "JAVA_OPT=%JAVA_OPT% -XX:+AlwaysPreTouch"
```

Batch file length: 2,208 lines: 42 Ln: 26 Col: 45 Sel: 13 | 1 Unix (LF) UTF-8 INS

③启动NameServer和Broker



```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19042.1415]
(c) Microsoft Corporation。保留所有权利。

D:\PROGRAMS\rocketmq-all-4.7.1-bin-release\bin>start mqnamesrv.cmd

D:\PROGRAMS\rocketmq-all-4.7.1-bin-release\bin>start mqbroker.cmd -n 127.0.0.1:9876 autoCreateTopicEnable=true

D:\PROGRAMS\rocketmq-all-4.7.1-bin-release\bin>
```

启动RocketMQ成功界面

```
C:\WINDOWS\system32\cmd.exe - mqnamesrv.cmd

Java HotSpot(TM) 64-Bit Server VM warning: Using the DefNew young collector with the CMS collector is deprecated and will likely be removed in a future release
Java HotSpot(TM) 64-Bit Server VM warning: UseCMSCompactAtFullCollection is deprecated and will likely be removed in a future release.
The Name Server boot success. serializeType=JSON
```

```
C:\WINDOWS\system32\cmd.exe - mqbroker.cmd -n 127.0.0.1:9876 autoCreateTopicEnable=true

The broker[DESKTOP-AJGQ8CS, 192.168.31.103:10911] boot success. serializeType=JSON and name server is 127.0.0.1:9876
```



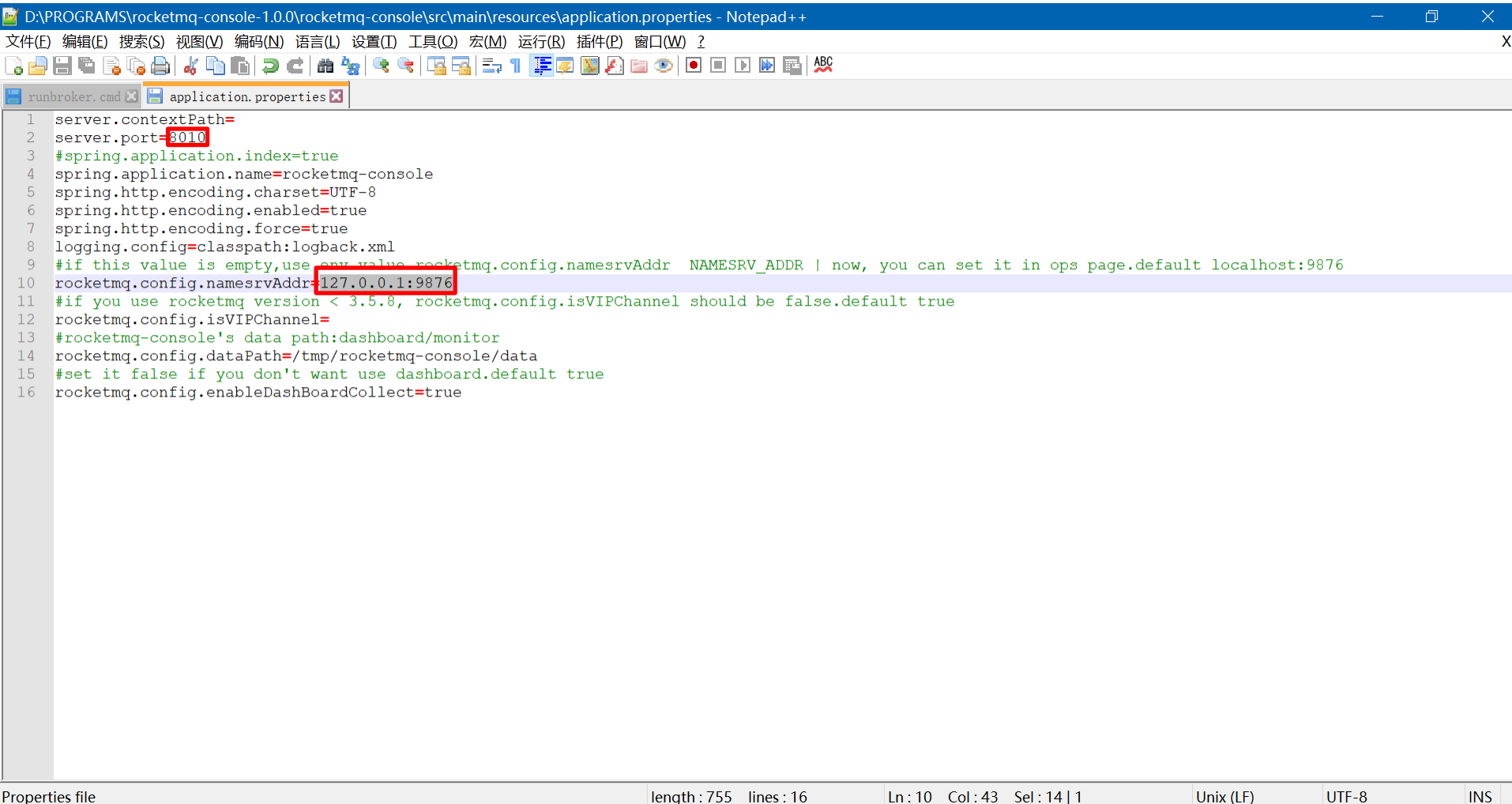
3.安装RocketMQ控制台

■安装RocketMQ控制台步骤:

- ✧①编辑dashboard项目配置文件
- ✧②打包和部署dashboard项目
- ✧③访问dashboard



①编辑dashboard项目配置文件



```
D:\PROGRAMS\rocketmq-console-1.0.0\rocketmq-console\src\main\resources\application.properties - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(I) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
runbroker.cmd application.properties
1 server.contextPath=
2 server.port=3010
3 #spring.application.index=true
4 spring.application.name=rocketmq-console
5 spring.http.encoding.charset=UTF-8
6 spring.http.encoding.enabled=true
7 spring.http.encoding.force=true
8 logging.config=classpath:logback.xml
9 #if this value is empty, use any value rocketmq.config.namesrvAddr NAMESRV_ADDR | now, you can set it in ops page.default localhost:9876
10 rocketmq.config.namesrvAddr=127.0.0.1:9876
11 #if you use rocketmq version < 3.5.8, rocketmq.config.isVIPChannel should be false.default true
12 rocketmq.config.isVIPChannel=
13 #rocketmq-console's data path:dashboard/monitor
14 rocketmq.config.dataPath=/tmp/rocketmq-console/data
15 #set it false if you don't want use dashboard.default true
16 rocketmq.config.enableDashBoardCollect=true
```

Properties file

length: 755 lines: 16

Ln: 10 Col: 43 Sel: 14 | 1

Unix (LF)

UTF-8

INS

②打包和部署dashboard项目

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19042.1415]
(c) Microsoft Corporation。保留所有权利。

D:\PROGRAMS\rocketmq-console-1.0.0\rocketmq-console>mvn clean package -Dmaven.test.skip=true_
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19042.1415]
(c) Microsoft Corporation。保留所有权利。

D:\PROGRAMS\rocketmq-console-1.0.0\rocketmq-console\target>java -jar rocketmq-console-ng-1.0.0.jar
```



③访问dashboard

