

RocketMQ

学习笔记

寿命齿轮

2024 年 1 月 20 日

作者的话

本文用于记录作者在学习 RocketMQ 过程中所记录的笔记。

学习资料来源：

1. RocketMQ 官网
2. 编程导航星球知识库：Yes 大佬的消息队列专栏
3. 网上各类与消息队列相关的博客

环境：

系统 Ubuntu22.04(云服务器)

Java 版本 17

目录

第一章 初识：消息队列	1
1.1 介绍	1
1.2 作用与优点	1
1.3 适用场景	2
1.3.1 异步场景举例：用户注册	2
1.3.2 解耦场景举例：订单-库存管理	2
1.3.3 削峰场景举例：秒杀活动	3
1.3.4 日志处理场景	3
1.3.5 消息通讯场景	4
1.4 常用消息队列框架	4
第二章 启动：RocketMQ	6
2.1 下载二进制文件包	6
2.2 启动 NameServer	7
2.3 启动 Broker+Proxy	9
第三章 领域模型	11
3.1 架构总览	11
3.2 基本概念	12
3.2.1 消息的生命周期	12
3.2.2 通信方式	13
3.2.3 消息传输模型	14
3.3 主题（Topic）	17
3.3.1 模型关系：主题	17

3.3.2	内部属性	17
3.3.3	Admin 工具相关操作	18
3.3.4	使用建议	18
3.4	队列 (MessageQueue)	18
3.4.1	模型关系：队列	19
3.4.2	内部属性	19
3.4.3	使用建议	20

第一章 初识：消息队列

1.1 介绍

消息队列，顾名思义，存放消息（可类比为请求）的队列（一种先进先出的数据结构）。

其是一种常用于分布式系统的中间件，可以在不同的应用程序、服务或系统之间传递消息，并且常用于解耦合不同部分的系统，使得系统更加可扩展和灵活。

基本原理：发送者将消息放入队列，接收者从队列中获取消息并处理。

消息队列实质是一种方式，一种在不同组件之间传递消息的通信方式。发送者和接收者之间不需要直接通信，它们只需了解如何发送和接收消息即可。

1.2 作用与优点

由上述内容，可推断出消息队列的一些作用：

- **解耦：**发送者和接收者只需要关心发送消息和接受消息，不用关心彼此。
- **异步：**发送者不关心消息的处理，即不用等待消息的响应，故支持异步。
- **削锋：**某些活动的流量过大、请求过多，可能导致系统宕机；消息队列可以作为缓冲区，将这些请求暂时存储起来，以避免瞬时高流量，然后按照系统处理能力逐步消费，实现流量的平滑处理，从而降低系统的压力，避免宕机。

以及身为分布式系统的固有优点：

- **可扩展性：**在解耦后，可方便地单独对发送者或接收者或消息队列进行动态伸缩。
- **可靠性：**由于消息队列允许多个消费者和生产者，并且通常支持消息持久化和复制，因此即使其中一个组件出现故障，系统仍然可以继续运行并且消息也不会丢失。

1.3 适用场景

（真实适用场景还是需要多实践才能掌握，这里仅介绍一些常用场景）

1.3.1 异步场景举例：用户注册

需求

用户注册后需向其发送注册邮件和注册短信。

设计

用户注册后，将注册信息写入数据库；发送注册邮件；发送短信。

如不使用消息队列，不进行异步解耦，即注册服务器需要同步远程调用写入数据库、发送注册邮件、发送短信的三个函数，将与其他应用发生多次交互，同时还得等待响应，假设一个操作需要 0.5s，则该操作会占用注册服务器一个线程的 1.5s。

使用消息队列后，注册服务器直接向消息队列中写入三个消息（数据库写入消息、邮件发送消息、短信发送消息），并且是异步发送不用等待返回，假设一次发送消息为 0.1s，也仅需 0.3s。

1.3.2 解耦场景举例：订单-库存管理

需求

用户下订单后，库存系统需要减少相对数量。

设计

用户下单后，订单系统需要通知库存系统。

详细设计

原设计：订单系统调用库存系统的接口。存在缺陷：假如库存系统无法访问，则订单减库存将失败，从而导致订单失败；订单系统依赖库存系统接口，存在耦合。

改进后：订单系统发送订单消息（用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功），库存系统读取订单消息并自行处理（订阅订单消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作）。解决缺陷：假如库存

系统无法访问，订单系统仅需要发送消息，可保持运转；订单消息仅发送消息，消息解读由库存系统进行（发布-订阅或消息队列模式），降低耦合度。

1.3.3 削峰场景举例：秒杀活动

需求

在秒杀活动中，大量用户同时抢购商品，可能会导致系统压力激增。为了应对这一情况，需要一种机制来平稳处理激增的请求流量，避免系统崩溃或性能下降。

设计

传统的处理方式可能会导致系统崩溃或性能下降。为了解决这个问题，可以使用消息队列来削峰填谷。

详细设计

1. 秒杀活动开始：当秒杀活动开始时，用户可以提交秒杀请求。
2. 请求入队：订单系统接收到用户的秒杀请求后，将请求消息写入消息队列，而不是立即处理。
3. 消息处理：秒杀请求消息被消息队列按照一定的规则（如先进先出）分发给后端处理程序。
4. 后端处理：后端处理程序逐条处理消息，检查库存并进行相应的处理（如减少库存、生成订单等）。

以此消息队列可平滑处理激增的请求流量，避免系统因突发流量而崩溃。

1.3.4 日志处理场景

需求

需要一种解决大量日志传输和实时处理的方案，以便对日志数据进行分析 and 可视化展示。

设计

设计一个分布式日志处理系统，包括以下组件：

1. 日志采集客户端：负责从各个日志源采集日志数据，并将数据定期写入消息队列中。
2. 消息队列：接收来自日志采集客户端的日志数据，负责数据的存储和转发。

3. 日志处理应用：订阅并消费 Kafka 队列中的日志数据，进行实时处理和分析。
4. Logstash：作为日志处理应用的一部分，负责对原始日志进行解析和转换，统一输出为 JSON 格式的数据。
5. Elasticsearch：作为日志处理应用的核心数据存储服务，接收 Logstash 处理后的 JSON 格式日志数据，实现实时的数据索引和查询。
6. Kibana：基于 Elasticsearch 的数据可视化组件，用于将 Elasticsearch 中的数据进行可视化展示和分析。

1.3.5 消息通讯场景

需求

需要一种高效的消息通讯机制，可以用于点对点通讯或者创建聊天室等场景，以实现实时的消息传递和交流。

设计

设计一个基于消息队列的消息通讯系统，包括以下两种场景：

1. 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；消息队列负责接收和转发客户端 A 和客户端 B 的消息。
2. 客户端 A、客户端 B 等多个客户端订阅同一主题：当有客户端发布消息时，消息队列将消息广播给所有订阅了该主题的客户端，客户端收到消息后进行展示。

1.4 常用消息队列框架

1. **RabbitMQ**：RabbitMQ 是一个开源的消息队列系统，实现了高级消息队列协议（AMQP），它是一个可靠、高可用、可扩展的消息代理。RabbitMQ 提供了多种消息传递模式，如点对点、发布/订阅等，适用于各种场景的应用程序。
2. **RocketMQ**：RocketMQ 是阿里巴巴开源的分布式消息队列系统，具有高吞吐量、低延迟、高可用性等特点。它支持丰富的消息模型，包括顺序消息、事务消息等，适用于大规模分布式系统的消息通信。
3. **Kafka**：Kafka 是由 Apache 软件基金会开发的分布式流处理平台和消息队列系统。Kafka 设计用于支持大规模的消息处理，具有高吞吐量、持久性、分区等特点，广泛应用于大数据领域。

4. **ActiveMQ**: ActiveMQ 是一个开源的消息中间件，实现了 Java Message Service (JMS) 规范。它支持多种传输协议，如 TCP、UDP、SSL 等，提供了丰富的功能，包括消息持久化、事务支持等。
5. **Amazon SQS**: Amazon SQS (Simple Queue Service) 是亚马逊提供的消息队列服务，可帮助构建分布式应用程序。它具有高可用性、可扩展性、灵活性等特点，适用于构建在亚马逊云平台上的应用程序。

本文将使用 RabbitMQ。

第二章 启动：RocketMQ

2.1 下载二进制文件包

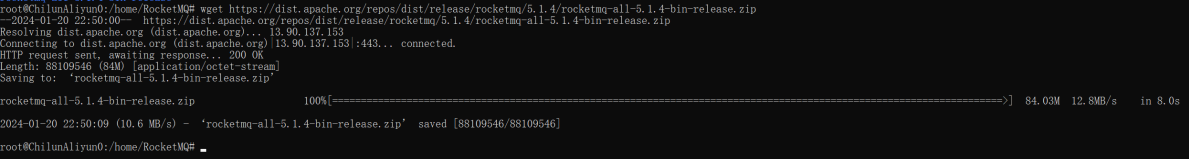
官网地址：<https://rocketmq.apache.org/zh/docs/quickStart/01quickstart>

获得二进制压缩包下载地址：

<https://dist.apache.org/repos/dist/release/rocketmq/5.1.4/rocketmq-all-5.1.4-bin-release.zip>

使用 `wget` 命令下载压缩包：

`wget https://dist.apache.org/repos/dist/release/rocketmq/5.1.4/rocketmq-all-5.1.4-bin-release.zip`



```
root@ChilunAliyun0:/home/RocketMQ# wget https://dist.apache.org/repos/dist/release/rocketmq/5.1.4/rocketmq-all-5.1.4-bin-release.zip
--2024-01-20 22:50:00-- https://dist.apache.org/repos/dist/release/rocketmq/5.1.4/rocketmq-all-5.1.4-bin-release.zip
Resolving dist.apache.org (dist.apache.org)... 13.90.137.153
Connecting to dist.apache.org (dist.apache.org) |13.90.137.153|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 88109546 (84M) [application/octet-stream]
Saving to: 'rocketmq-all-5.1.4-bin-release.zip'

rocketmq-all-5.1.4-bin-release.zip 100%[=====] 84.03M 12.5MB/s in 8.0s

2024-01-20 22:50:09 (10.6 MB/s) - 'rocketmq-all-5.1.4-bin-release.zip' saved [88109546/88109546]

root@ChilunAliyun0:/home/RocketMQ#
```

图 2.1: 下载二进制压缩包

使用 `unzip` 命令解压二进制文件压缩包：

`unzip rocketmq-all-5.1.4-bin-release.zip`

```
root@ChilunAliyun0:/home/RocketMQ# ls
rocketmq-all-5.1.4-bin-release.zip
root@ChilunAliyun0:/home/RocketMQ# unzip rocketmq-all-5.1.4-bin-release.zip
Archive:  rocketmq-all-5.1.4-bin-release.zip
  inflating: rocketmq-all-5.1.4-bin-release/LICENSE
  inflating: rocketmq-all-5.1.4-bin-release/NOTICE
  inflating: rocketmq-all-5.1.4-bin-release/README.md
   creating: rocketmq-all-5.1.4-bin-release/benchmark/
  inflating: rocketmq-all-5.1.4-bin-release/benchmark/consumer.sh
  inflating: rocketmq-all-5.1.4-bin-release/benchmark/runclass.sh
  inflating: rocketmq-all-5.1.4-bin-release/benchmark/tproducer.sh
  inflating: rocketmq-all-5.1.4-bin-release/benchmark/batchproducer.sh
  inflating: rocketmq-all-5.1.4-bin-release/benchmark/producer.sh
  inflating: rocketmq-all-5.1.4-bin-release/benchmark/shutdown.sh
   creating: rocketmq-all-5.1.4-bin-release/bin/
  inflating: rocketmq-all-5.1.4-bin-release/bin/tools.cmd
  inflating: rocketmq-all-5.1.4-bin-release/bin/runserver.sh
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqbroker.numanode0
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqproxy.cmd
  inflating: rocketmq-all-5.1.4-bin-release/bin/runbroker.cmd
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqbroker.numanodel
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqbrokercontainer
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqproxy
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqshutdown.cmd
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqcontroller
  inflating: rocketmq-all-5.1.4-bin-release/bin/runserver.cmd
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqadmin
  inflating: rocketmq-all-5.1.4-bin-release/bin/mqadmin.cmd
   creating: rocketmq-all-5.1.4-bin-release/bin/controller/
```

图 2.2: 解压二进制文件

2.2 启动 NameServer

进入目录 `rocketmq-all-5.1.4-bin-release`，执行命令：

`nohup sh bin/mqnamesrv &`

命令讲解：

- **nohup**：这代表“不挂起”。在终端中执行命令然后关闭终端时，与该命令相关联的进程通常也会终止。nohup 可以防止这种情况发生。
- **sh**：执行脚本文件的 shell 命令。
- **bin/mqnamesrv**：要运行的脚本路径。
- **&**：后台运行。

发现运行失败：

```
root@ChilunAliyun0:/home/RocketMQ/rocketmq-all-5.1.4-bin-release# nohup sh bin/mqnamesrv &
[1] 107550
root@ChilunAliyun0:/home/RocketMQ/rocketmq-all-5.1.4-bin-release# nohup: ignoring input and app
ending output to 'nohup.out'

[1]+  Exit 1                  nohup sh bin/mqnamesrv
root@ChilunAliyun0:/home/RocketMQ/rocketmq-all-5.1.4-bin-release#
```

图 2.3: 名字服务器启动失败

查看 nohup.out 文件, 发现报错: **OpenJDK 64-Bit Server VM warning: INFO: os::commit_memory(0x0000000700000000, 4294967296, 0) failed; error= 'Not enough space' (errno=12)**

操作系统内存不足 (由于 RocketMQ 对内存要求极高, 所以自己用云服务器运行基本都会报错), 进行修改:

进入 rocketmq-all-5.1.4-bin-release/bin 目录, 对 runserver.sh 和 runbroker.sh 以及 tools.sh 进行修改。(可使用 vim 的/+ 关键字进行查找)

1. runserver.sh:

```
JAVA_OPT="$JAVA_OPT -server -Xms4g -Xmx4g -Xmn2g -XX:MetaspaceSize=128m  
-XX:MaxMetaspaceSize=320m"
```

替换为

```
JAVA_OPT="$JAVA_OPT -server -Xms256m -Xmx256m -Xmn128m  
-XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m"
```

注意有两处。

2. runbroker.sh:

```
JAVA_OPT="$JAVA_OPT -server -Xms8g -Xmx8g"
```

替换为 **JAVA_OPT="\$JAVA_OPT -server -Xms256m -Xmx256m",**

```
JAVA_OPT="$JAVA_OPT -Xmn8G -XX:+UseConcMarkSweepGC
```

替换为

```
JAVA_OPT="$JAVA_OPT -Xmn256m -XX:+UseConcMarkSweepGC
```

3. tools.sh:

```
JAVA_OPT="$JAVA_OPT -server -Xms1g -Xmx1g -Xmn256m -XX:MetaspaceSize=128m  
-XX:MaxMetaspaceSize=128m"
```

替换为

```
JAVA_OPT="$JAVA_OPT -server -Xms256g -Xmx256g -Xmn128m  
-XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m"。
```

再次输入命令: **nohup sh bin/mqnamesrv &**

未报错, 查看 nohup.out 文件, 发现启动成功:

```
#
# There is insufficient memory for the Java Runtime Environment to continue.
# Native memory allocation (mmap) failed to map 4294967296 bytes for committing reserved memory
#
# An error report file with more information is saved as:
# /home/RocketMQ/rocketmq-all-5.1.4-bin-release/hs_err_pid107578.log
The Name Server boot success. serializeType=JSON, address 0.0.0.0:9876
```

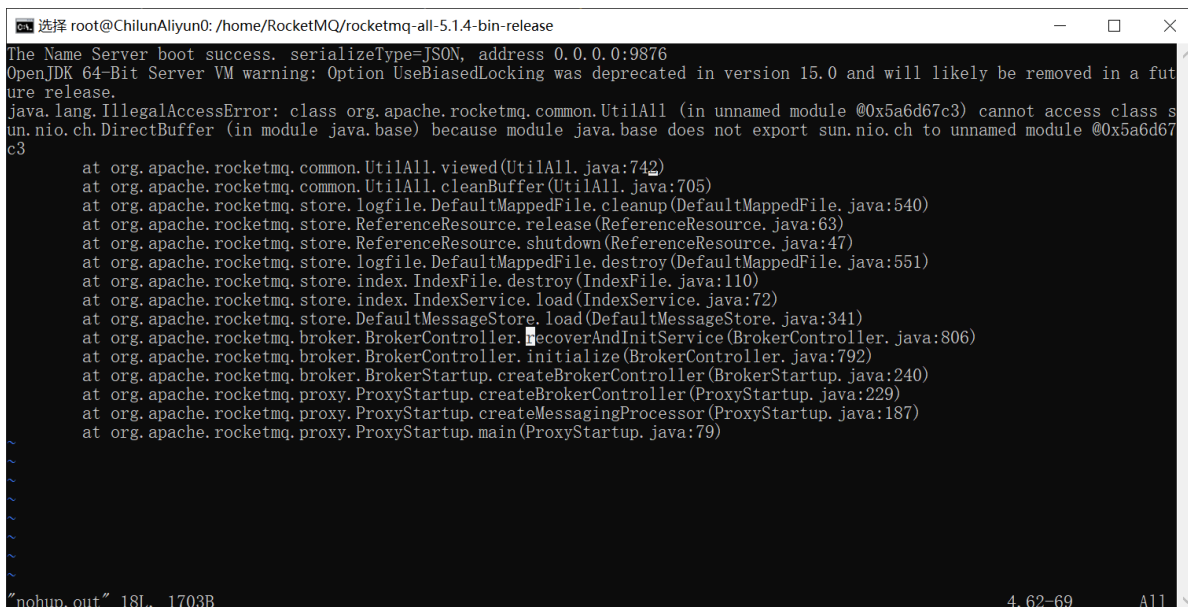
图 2.4: 名称服务器启动成功

2.3 启动 Broker+Proxy

执行命令:

```
nohup sh bin/mqbroker -n localhost:9876 --enable-proxy &
```

未报错, 查看 nohup.out 文件, 发现启动还是失败:



```
选择 root@ChilunAliyun0: /home/RocketMQ/rocketmq-all-5.1.4-bin-release
The Name Server boot success. serializeType=JSON, address 0.0.0.0:9876
OpenJDK 64-Bit Server VM warning: Option UseBiasedLocking was deprecated in version 15.0 and will likely be removed in a future release.
java.lang.IllegalAccessError: class org.apache.rocketmq.common.UtilAll (in unnamed module @0x5a6d67c3) cannot access class sun.nio.ch.DirectBuffer (in module java.base) because module java.base does not export sun.nio.ch to unnamed module @0x5a6d67c3
    at org.apache.rocketmq.common.UtilAll.viewed(UtilAll.java:742)
    at org.apache.rocketmq.common.UtilAll.cleanBuffer(UtilAll.java:705)
    at org.apache.rocketmq.store.logfile.DefaultMappedFile.cleanup(DefaultMappedFile.java:540)
    at org.apache.rocketmq.store.ReferenceResource.release(ReferenceResource.java:63)
    at org.apache.rocketmq.store.ReferenceResource.shutdown(ReferenceResource.java:47)
    at org.apache.rocketmq.store.logfile.DefaultMappedFile.destroy(DefaultMappedFile.java:551)
    at org.apache.rocketmq.store.index.IndexFile.destroy(IndexFile.java:110)
    at org.apache.rocketmq.store.index.IndexService.load(IndexService.java:72)
    at org.apache.rocketmq.store.DefaultMessageStore.load(DefaultMessageStore.java:341)
    at org.apache.rocketmq.broker.BrokerController.recoverAndInitService(BrokerController.java:806)
    at org.apache.rocketmq.broker.BrokerController.initialize(BrokerController.java:792)
    at org.apache.rocketmq.broker.BrokerStartup.createBrokerController(BrokerStartup.java:240)
    at org.apache.rocketmq.proxy.ProxyStartup.createBrokerController(ProxyStartup.java:229)
    at org.apache.rocketmq.proxy.ProxyStartup.createMessagingProcessor(ProxyStartup.java:187)
    at org.apache.rocketmq.proxy.ProxyStartup.main(ProxyStartup.java:79)
"nohup.out" 18L, 1703B 4,62-69 All
```

图 2.5: broker 启动失败

原因是 JAVA 版本过高, 进行修复。

修改 runbroker.sh 文件:

在 `numactl --interleave=all pwd > /dev/null 2>&1` 上方添加

```
$JAVA $JAVA_OPT --add-exports=java.base/sun.nio.ch=ALL-UNNAMED $@
```

然后再次运行 `nohup sh bin/mqbroker -n localhost:9876 --enable-proxy &`

并查看 nohup.out, 发现为不断更新的日志文件, 推测运行成功。

查看 /root/logs/rocketmqlogs, 发现运行成功。

```

root@ChilunAliyun0: ~/logs/rocketmqlogs
2024-01-20 23:55:14 INFO main - Try to start service thread:QueueLockManager started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:QueueLockManager started:true lastThread:Thread[QueueLockManager, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopRevokeService_0 started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopRevokeService_0 started:true lastThread:Thread[PopRevokeService_0, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopRevokeService_1 started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopRevokeService_1 started:true lastThread:Thread[PopRevokeService_1, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopRevokeService_2 started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopRevokeService_2 started:true lastThread:Thread[PopRevokeService_2, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopRevokeService_3 started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopRevokeService_3 started:true lastThread:Thread[PopRevokeService_3, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopRevokeService_4 started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopRevokeService_4 started:true lastThread:Thread[PopRevokeService_4, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopRevokeService_5 started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopRevokeService_5 started:true lastThread:Thread[PopRevokeService_5, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopRevokeService_6 started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopRevokeService_6 started:true lastThread:Thread[PopRevokeService_6, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopRevokeService_7 started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopRevokeService_7 started:true lastThread:Thread[PopRevokeService_7, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PopLongPollingService started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:PopLongPollingService started:true lastThread:Thread[PopLongPollingService, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:TopicQueueMappingCleanService started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:TopicQueueMappingCleanService started:true lastThread:Thread[TopicQueueMappingCleanService, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:FileWatchService started:false lastThread:null
2024-01-20 23:55:14 INFO TopicQueueMappingCleanService - Start topic queue mapping clean service thread!
2024-01-20 23:55:14 INFO main - Start service thread:FileWatchService started:true lastThread:Thread[FileWatchService, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:PullRequestHoldService started:false lastThread:null
2024-01-20 23:55:14 INFO FileWatchService - FileWatchService service started
2024-01-20 23:55:14 INFO main - Start service thread:PullRequestHoldService started:true lastThread:Thread[PullRequestHoldService, 5, main]
2024-01-20 23:55:14 INFO PullRequestHoldService - PullRequestHoldService service started
2024-01-20 23:55:14 INFO main - Try to start service thread:BroadcastOffsetManager started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:BroadcastOffsetManager started:true lastThread:Thread[BroadcastOffsetManager, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:ColdDataPullRequestHoldService started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:ColdDataPullRequestHoldService started:true lastThread:Thread[ColdDataPullRequestHoldService, 5, main]
2024-01-20 23:55:14 INFO main - Try to start service thread:ColdDataCgCtrService started:false lastThread:null
2024-01-20 23:55:14 INFO main - Start service thread:ColdDataCgCtrService started:true lastThread:Thread[ColdDataCgCtrService, 5, main]
2024-01-20 23:55:14 INFO main - ScheduleServiceStatus changed to true
2024-01-20 23:55:14 INFO main - load /root/store/config/delayOffset.json OK
2024-01-20 23:55:14 INFO main - TransactionCheckService status changed to true
2024-01-20 23:55:14 INFO main - TransactionCheckService started:false lastThread:null
2024-01-20 23:55:14 INFO main - Try to start service thread:TransactionalMessageCheckService started:true lastThread:Thread[TransactionalMessageCheckService, 5, main]
2024-01-20 23:55:14 INFO main - Set PopRevokeService Status to true
2024-01-20 23:55:14 INFO main - The broker [ChilunAliyun0, 172.18.187.94:10911] boot success, serialType=KVM and name server is localhost:9876
2024-01-20 23:55:23 INFO BrokerControllerScheduledThread1 - Dispatch task fall behind commit log Obytes
2024-01-20 23:55:24 INFO brokerOutApi_thread_2 - Registering current broker to name server completed. TargetHost=localhost:9876
2024-01-20 23:55:24 INFO brokerOutApi_thread_3 - Registering current broker to name server completed. TargetHost=localhost:9876
2024-01-20 23:55:24 INFO BrokerControllerScheduledThread1 - Dispatch task fall behind commit log Obytes
2024-01-20 23:55:24 INFO brokerOutApi_thread_4 - Registering current broker to name server completed. TargetHost=localhost:9876
2024-01-20 23:55:24 INFO brokerOutApi_thread_1 - Registering current broker to name server completed. TargetHost=localhost:9876
2024-01-20 23:55:24 INFO BrokerControllerScheduledThread1 - Dispatch task fall behind commit log Obytes
2024-01-20 23:55:24 INFO brokerOutApi_thread_2 - Registering current broker to name server completed. TargetHost=localhost:9876
2024-01-20 23:55:24 INFO brokerOutApi_thread_3 - Registering current broker to name server completed. TargetHost=localhost:9876
2024-01-20 23:55:24 INFO BrokerControllerScheduledThread1 - Dispatch task fall behind commit log Obytes
2024-01-20 23:55:24 INFO brokerOutApi_thread_4 - Registering current broker to name server completed. TargetHost=localhost:9876
2024-01-20 23:55:24 INFO brokerOutApi_thread_1 - Registering current broker to name server completed. TargetHost=localhost:9876
/ success

```

图 2.6: broker 启动成功

至此, RocketMQ 启动成功。

第三章 领域模型

3.1 架构总览

组件示意图：

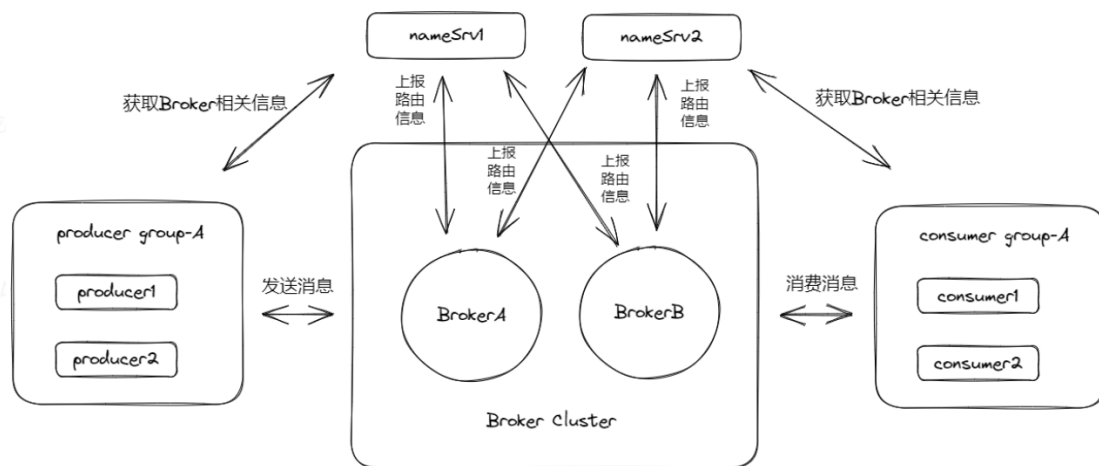


图 3.1: RocketMQ 全局图

- **Producer**：消息生产者。
- **NameSrv**：名称服务，即路由注册中心，可记录中转者提供给消费者和生产者。
- **Broker**：中转者（也可以认为是队列 Queue），负责接受、存储、转发消息。
- **Consumer**：消息消费者。
- **Producer group**：生产者组。
- **Consumer group**：消费者组。
- **Broker cluster**：中转者集群。

3.2 基本概念

3.2.1 消息的生命周期

RocketMQ 中消息的生命周期主要分为**消息生产**、**消息存储**、**消息消费**这三部分。

生产者生产消息并发送至 RocketMQ 服务端，消息被存储在服务端的主题中，消费者通过订阅主题消费消息。

消息生产

生产者 (Producer)：在 RocketMQ 中，生产者是用来发送消息的组件，通常它被集成到业务系统的前端。生产者是轻量级匿名无身份的。

消息存储

消息 (Message)：RocketMQ 的最小传输单元。消息具备不可变性，在初始化发送和完成存储后即不可变。

主题 (Topic)：RocketMQ 消息传输和存储的分组容器，主题内部由多个队列组成，消息的存储和水平扩展实际是通过主题内的队列实现的。

队列 (MessageQueue)：RocketMQ 消息传输和存储的实际单元容器，类比于其他消息队列中的分区。RocketMQ 使用流式特性的无限队列结构来存储消息，消息在队列内会按顺序进行存储。

消息消费

消费者分组 (ConsumerGroup)：RocketMQ 发布订阅模型中定义的独立的消费身份分组，用于统一管理底层运行的多个消费者 (Consumer)。同一个消费组的多个消费者必须保持消费逻辑和配置一致，共同分担该消费组订阅的消息，实现消费能力的水平扩展。

消费者 (Consumer)：RocketMQ 消费消息的运行实体，通常它被集成到业务系统的后端。消费者必须被指定到某一个消费组中。

订阅关系 (Subscription)：RocketMQ 的订阅关系是指在发布订阅模型中，用于配置消息过滤、重试以及消费进度的规则。订阅关系是以消费组为单位进行管理的，消费组通过定义订阅关系来控制该组下的消费者如何处理消息的过滤、重试以及消费进度恢复等操作。简单来说，订阅关系就是消费组对消息的一种规则设定，可以让我们更加灵活地控制消息的处理方式，确保系统能够按照我们预期的方式来消费和处理消息。

（RocketMQ 的订阅关系除过滤表达式之外都是持久化的，即服务端重启或请求断开，订阅关系依然保留。）

3.2.2 通信方式

分布式系统架构思想下，将复杂系统拆分为多个独立的子模块，例如微服务模块。此时就需要考虑子模块间的远程通信，典型的通信模式分为以下两种，一种是**同步的 RPC 远程调用**；一种是**基于中间件代理的异步通信**。

同步 RPC 调用模型

同步 RPC 调用模型下，不同系统之间直接进行调用通信，每个请求直接从调用方发送到被调用方，然后要求被调用方立即返回响应结果给调用方，以确定本次调用结果是否成功。

（注意：此处的同步并不代表 RPC 的编程接口方式，RPC 也可以有异步非阻塞调用的编程方式，但本质上仍然是需要在指定时间内得到目标端的直接响应。）

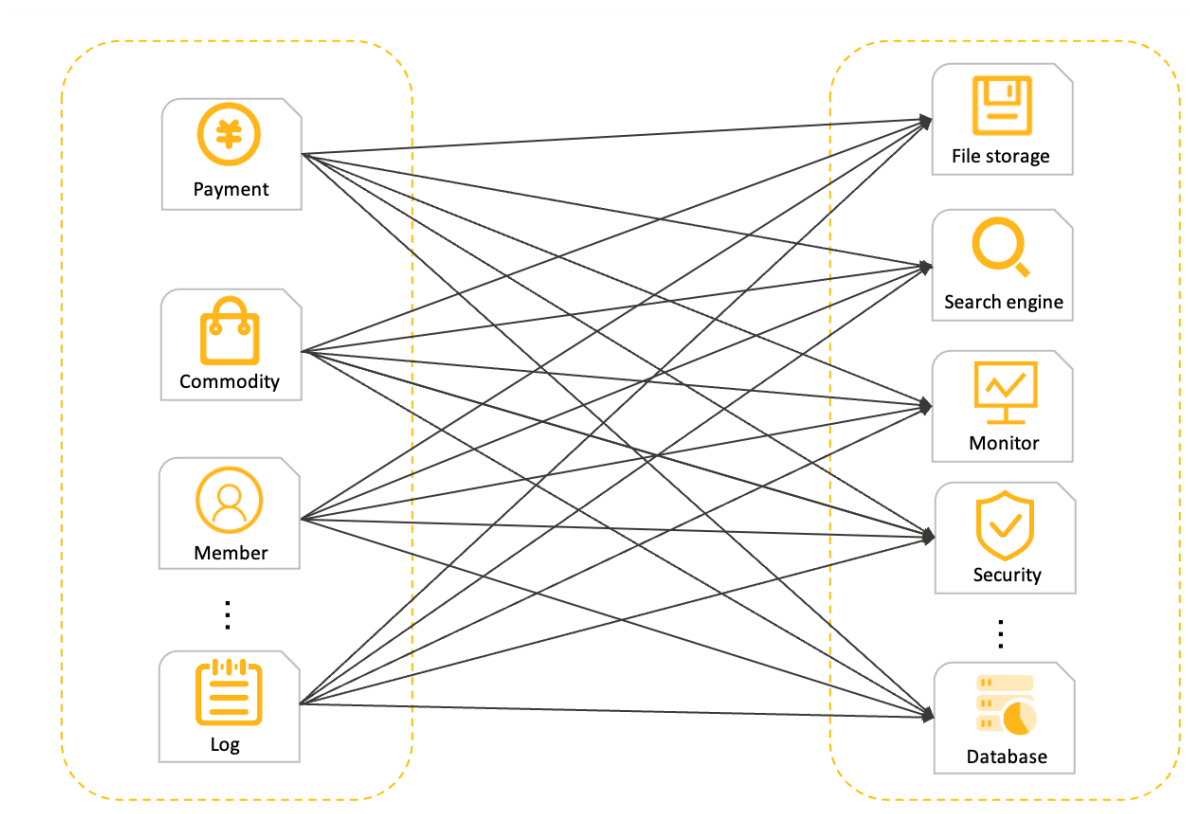


图 3.2: 同步 RPC 调用模型

异步通信模型

异步消息通信模式下，各子系统之间无需强耦合直接连接，调用方只需要将请求转化成异步事件（消息）发送给中间代理，发送成功即可认为该异步链路调用完成，剩下的工作中间代理会负责将事件可靠通知到下游的调用系统，确保任务执行完成。该中间代理一般就是消息中间件。

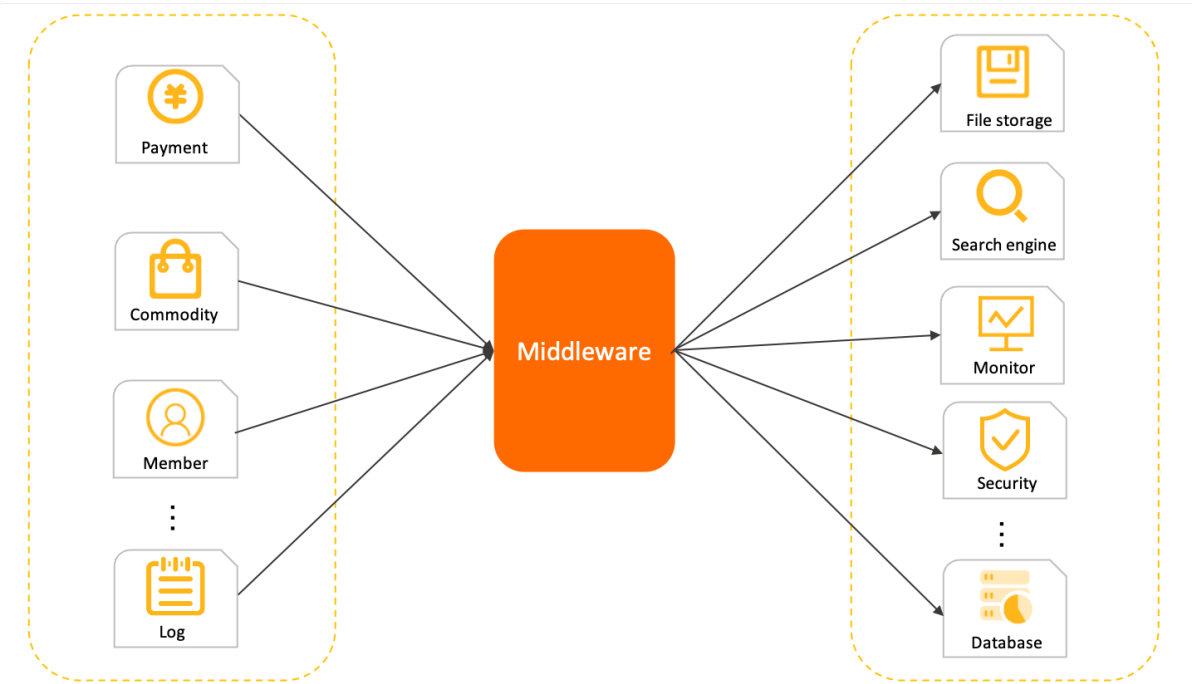


图 3.3: 异步通信模型

异步通信优势如下：

- 1. 星型结构系统，拓扑简单，易于维护和管理。
- 2. 上下游耦合性弱，可以独立升级和变更，不会互相影响。
- 3. 容量削峰填谷。基于消息的中间代理往往具备很强的流量缓冲和整形能力，业务流量高峰到来时不会击垮下游。

3.2.3 消息传输模型

主流的消息中间件的传输模型主要为点对点模型和发布订阅模型。

点对点模型

点对点模型也叫队列模型，具有如下特点：

1. 消费匿名：消息上下游沟通的唯一的身份就是队列，下游消费者从队列获取消息无法申明独立身份（即中间件对同一个队列中的所有消费者都一视同仁）。
2. 一对一通信：基于消费匿名特点，下游消费者即使有多个，但都没有自己独立的身份，因此共享队列中的消息，每一条消息都只会被唯一一个消费者处理。因此点对点模型只能实现一对一通信。

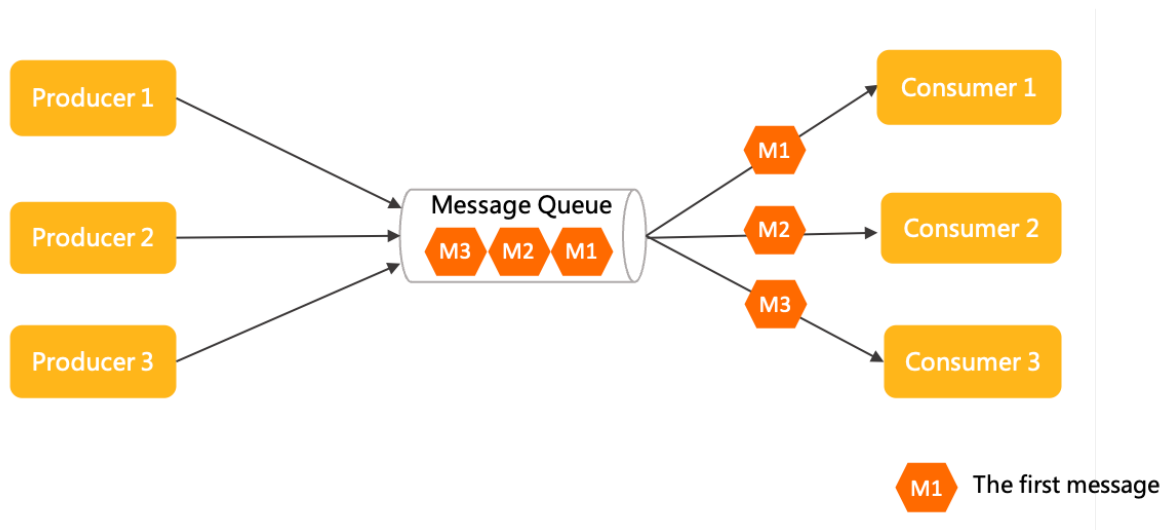


图 3.4: 点对点模型

发布订阅模型

发布订阅模型具有如下特点：

1. 消费独立：相比队列模型的匿名消费方式，发布订阅模型中消费方都会具备的身份，一般叫做订阅组（订阅关系），不同订阅组之间相互独立不会相互影响。
2. 一对多通信：基于独立身份的设计，同一个主题内的消息可以被多个订阅组处理，每个订阅组都可以拿到全量消息。因此发布订阅模型可以实现一对多通信。

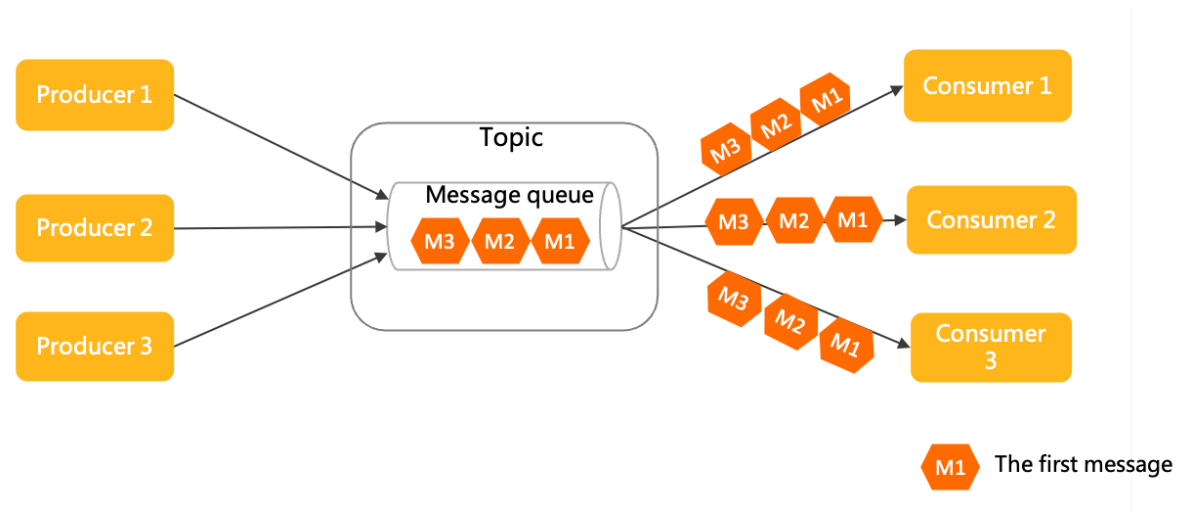


图 3.5: 发布订阅模型

对比

点对点模型和发布订阅模型各有优势，点对点模型更为简单，而发布订阅模型的扩展性更高。RocketMQ 使用的传输模型为发布订阅模型，因此也具有发布订阅模型的特点。

领域模型图

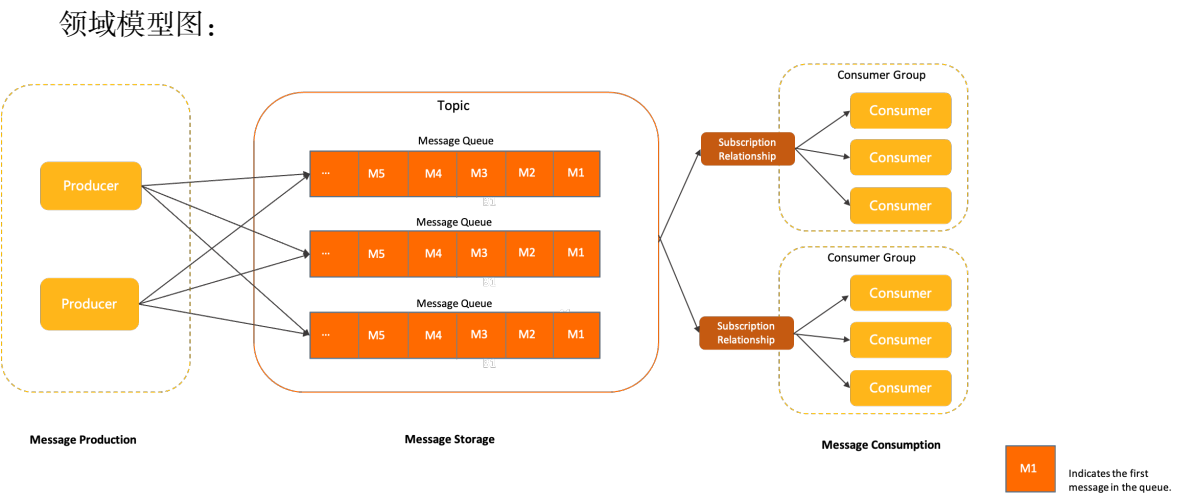


图 3.6: 领域模型图

3.3 主题 (Topic)

主题是 RocketMQ 中消息传输和存储的顶层容器，用于标识同一类业务逻辑的消息。主题的作用主要如下：

1. **数据分类**：在 RocketMQ 的方案设计中，建议将不同业务类型的数据拆分到不同的主题中管理，通过主题实现存储的隔离性和订阅隔离性。
2. **区分身份和权限**：RocketMQ 的消息本身是匿名无身份的，同一分类的消息使用相同的主题来做身份识别和权限管理。

3.3.1 模型关系：主题

在整个 RocketMQ 领域模型中，主题所处的位置如下（橙色区域）：

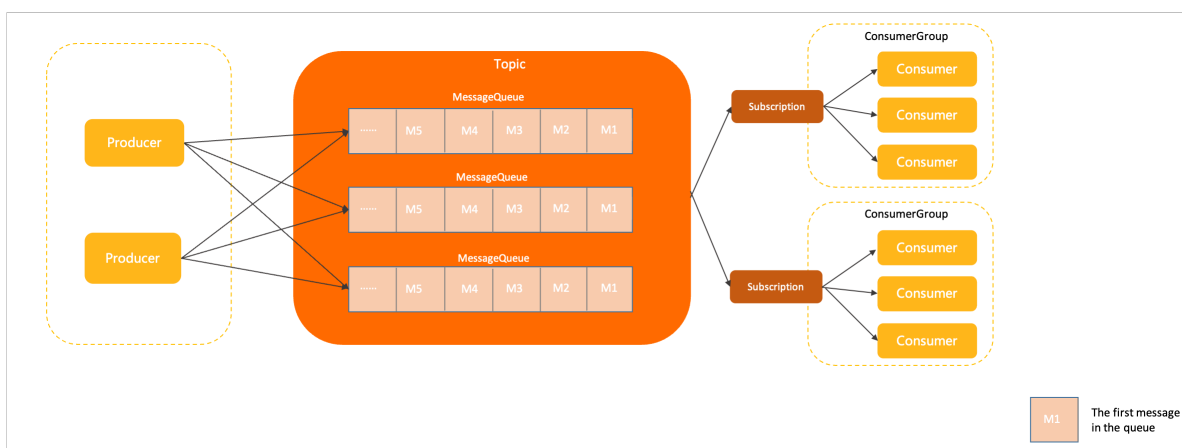


图 3.7: 模型关系：主题

主题是 RocketMQ 的顶层存储，所有消息资源的定义都在主题内部完成，但主题是一个**逻辑概念**，并不是实际的消息容器。

主题内部由多个队列组成，消息的存储和水平扩展能力最终是由队列实现的；并且针对主题的所有约束和属性设置，最终也是通过主题内部的队列来实现。

3.3.2 内部属性

1. **主题名称**：主题的名称，用于标识主题，主题名称集群内全局唯一，由用户创建主题时定义。
2. **队列列表**：队列作为主题的组成单元，是消息存储的实际容器，一个主题内包含一个或多个队列，消息实际存储在主题的各队列内；队列数量创建主题时定义（一个主题内至少包

含一个队列)。

3. 消息类型：主题所支持的消息类型（包括普通消息、顺序消息、定时/延时消息、事务消息），每个主题只允许发送一种消息类型的消息。

3.3.3 Admin 工具相关操作

创建主题操作命令：

```
sh mqadmin updateTopic -n <nameserver_address> -t <topic_name> -c <cluster_name> -a +message.type=<message_type>
```

各类操作（详情见RocketMQ 官网）：

1. updateTopic：创建/更新 Topic 配置。
2. deleteTopic：删除 Topic。
3. topicList：查看 Topic 列表信息。
4. topicRoute：查看 Topic 路由信息。
5. topicStatus：查看 Topic 消息队列 offset。
6. topicClusterList：查看 Topic 所在集群列表。
7. updateTopicPerm：更新 Topic 读写权限。
8. updateOrderConf：以平均负载算法计算消费者列表负载消息队列的负载结果。
9. statsAll：打印 Topic 订阅关系、TPS、积累量、24h 读写总量等信息。

3.3.4 使用建议

- 合理拆分主题，注意消息量级与消息时效性，避免将时效性要求高的业务消息与量级大的业务消息归到同一主题。
- 单一主题只收发一种类型消息，避免混用。
- 主题管理尽量避免自动化机制，并在生产环境需保证严格管理主题资源，不随意进行增、删、改、查操作。

3.4 队列（MessageQueue）

队列是 RocketMQ 中消息存储和传输的实际容器，也是 RocketMQ 消息的最小存储单元。

RocketMQ 的所有主题都是由多个队列组成，以此实现队列数量的水平拆分和队列内部的流式存储。

队列的作用如下：

- 1. 存储顺序性：队列天然具备顺序性，即消息按照进入队列的顺序写入存储。消息在队列中的位置通过位点（Offset）进行记录，队列头部为最早写入的消息（Offset=0），队列尾部为最新写入的消息（Offset 逐渐递增）。
- 2. 流式操作语义：RocketMQ 基于队列的存储模型可确保消息从任意位点（Offset）读取任意数量的消息，以此实现类似聚合读取、回溯读取等特性，这些特性是队列存储模型特有的。

3.4.1 模型关系：队列

在整个 RocketMQ 领域模型中，队列所处的位置如下（橙色区域）：

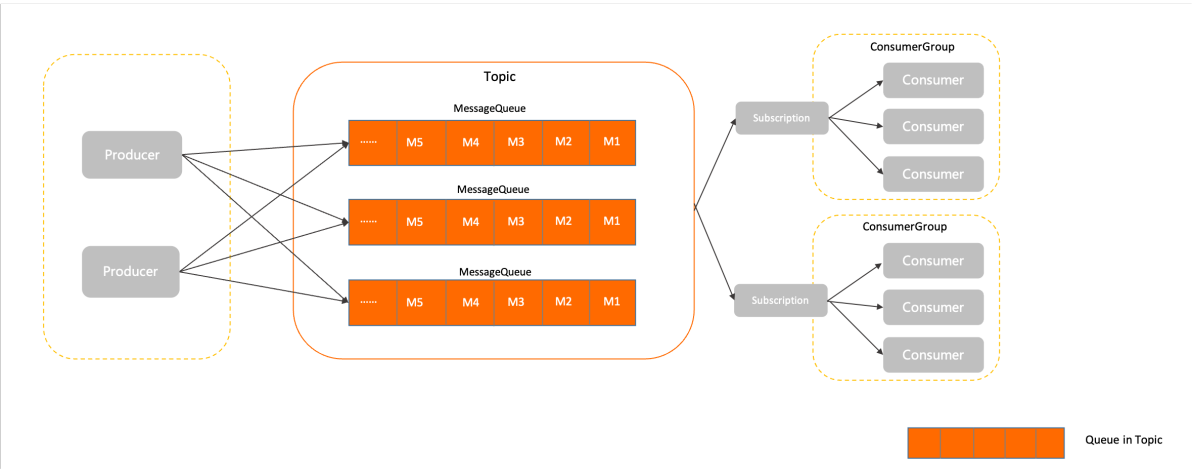


图 3.8: 模型关系：队列

RocketMQ 默认提供**可靠的消息存储机制**，所有发送成功的消息都被持久化存储到队列中，配合生产者和消费者客户端的调用保证至少投递一次。

队列属于主题的一部分，虽然所有的消息资源以主题粒度管理，但实际的操作实现是面向队列。例如，生产者指定某个主题，向主题内发送消息，但实际消息发送到该主题下的某个队列中。

RocketMQ 中可通过修改队列数量，以此实现横向的水平扩容和缩容。

3.4.2 内部属性

- 1. 读写权限：当前队列是否可以读写数据（队列的读写权限属于运维侧操作，不建议频繁修改）。

3.4.3 使用建议

- 队列数量的设置应遵循少用够用原则，否则可能导致集群元数据膨胀、系统负荷增加（队列越多，消费者的轮询越多）。
- 在集群水平扩容增加节点后，为了保证集群流量的负载均衡，建议在新的服务节点上新增队列，或将旧的队列迁移到新的服务节点上。
- 顺序消息的并发度会在一定程度上受队列数量的影响，当达到性能瓶颈时需要再增加队列。