第一章　领域模型

1.1　架构总览

组件示意图：

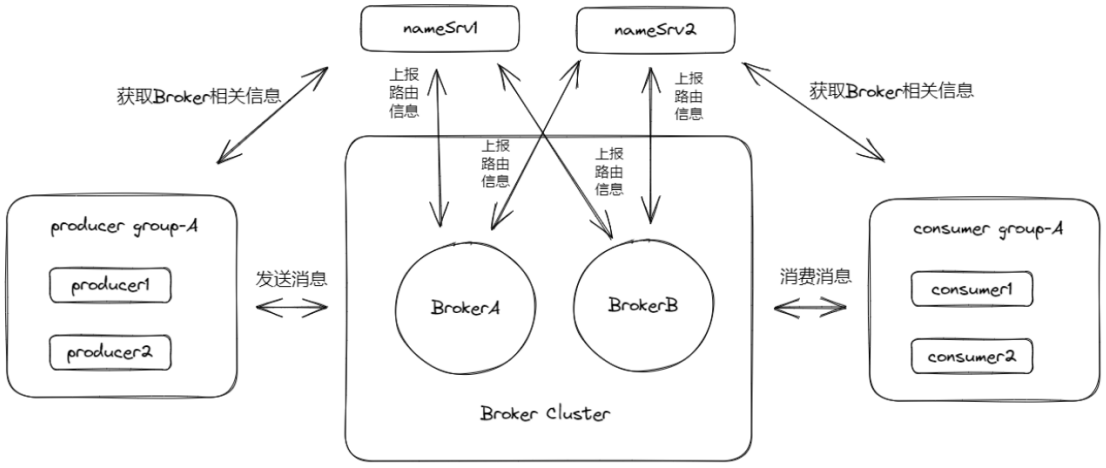

图 1.1: RocketMQ 全局图

- **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 Queen），负责接受、存储、转发消息。
- **Comsumer**：消息消费者。
- Producer group：生产者组。
- Comsumer group：消费者组。
- Broker cluster：中转者集群。

1.2　基本概念

1.2.1　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。）

1.2.2　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。）

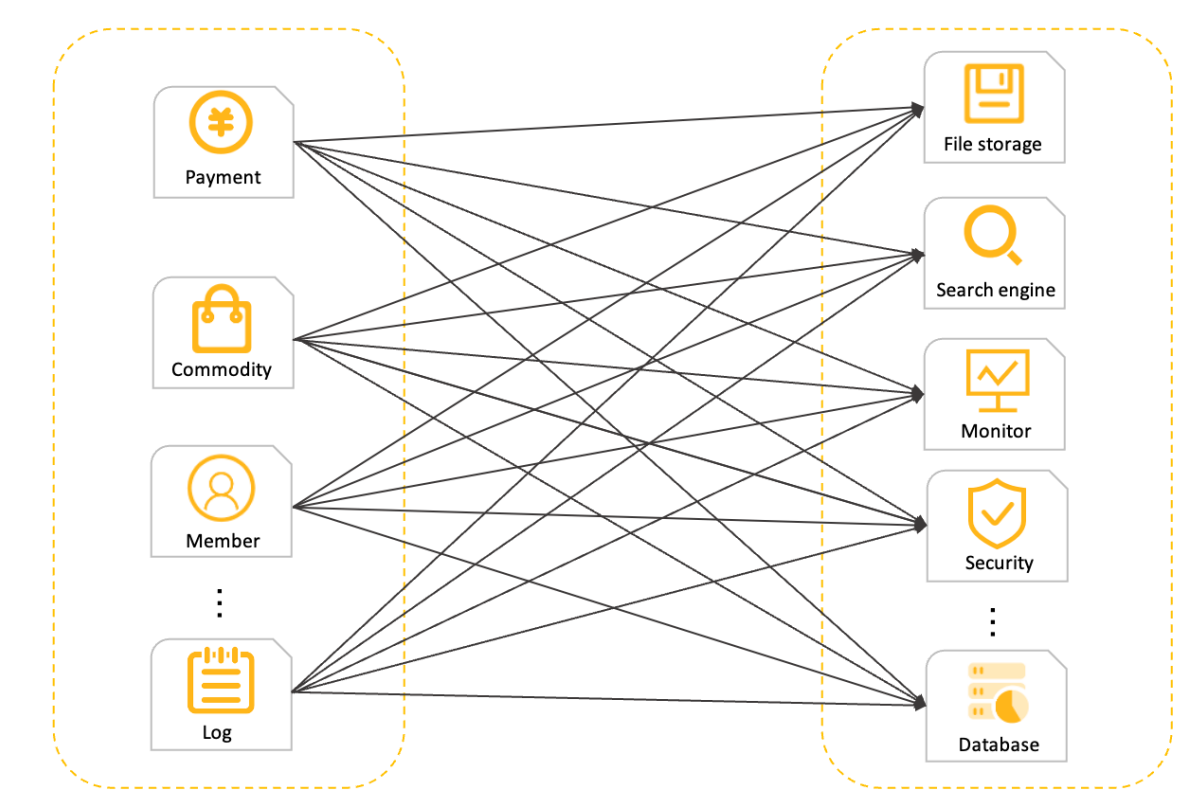

图 1.2: 同步 RPC 调用模型

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。

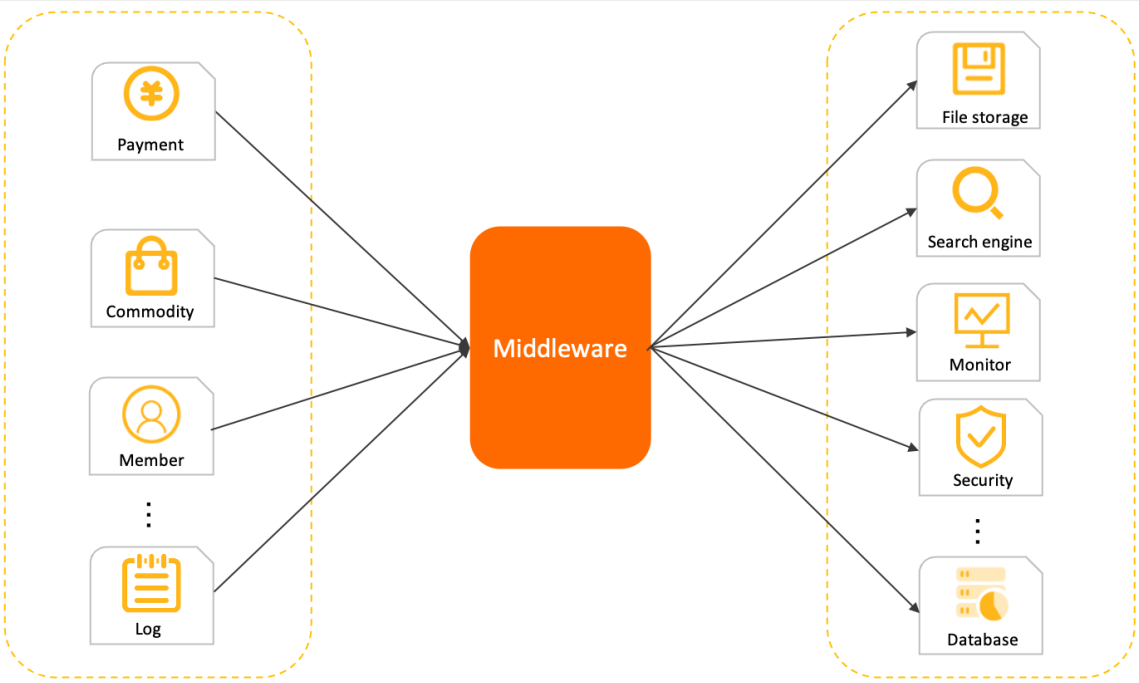

图 1.3: 异步通信模型

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。

1.2.3　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。

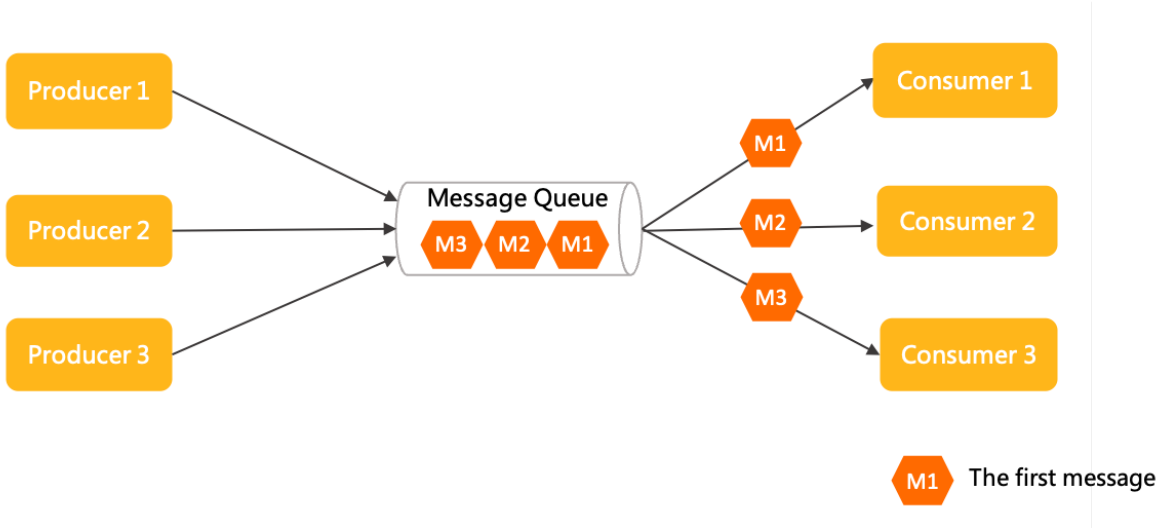

图 1.4: 点对点模型

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。

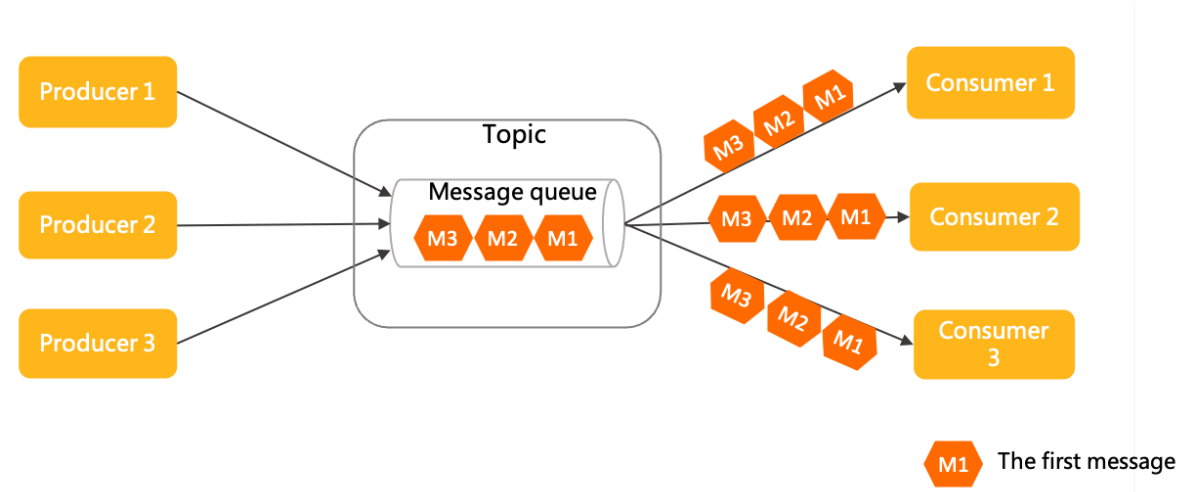

图 1.5: 发布订阅模型

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

领域模型图：

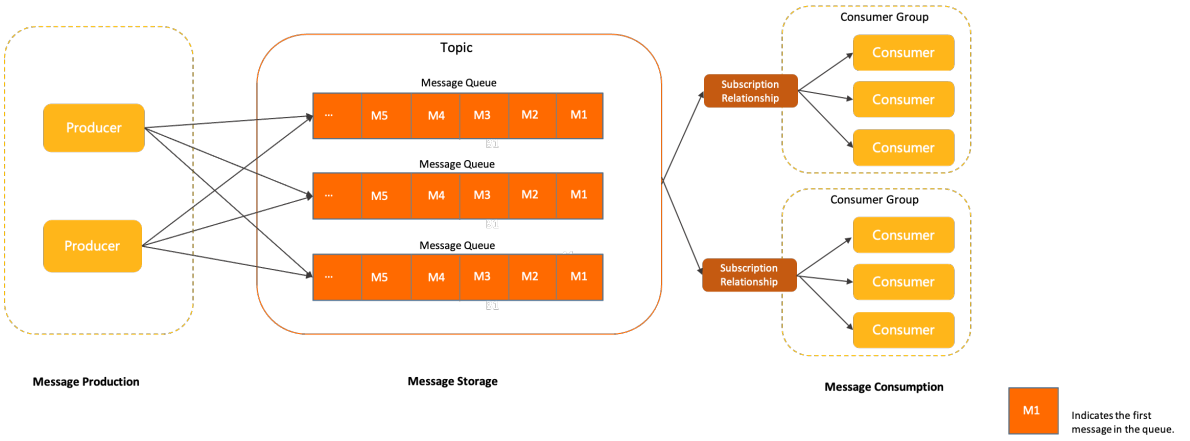

图 1.6: 领域模型图

1.3　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。

1.3.1　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）：

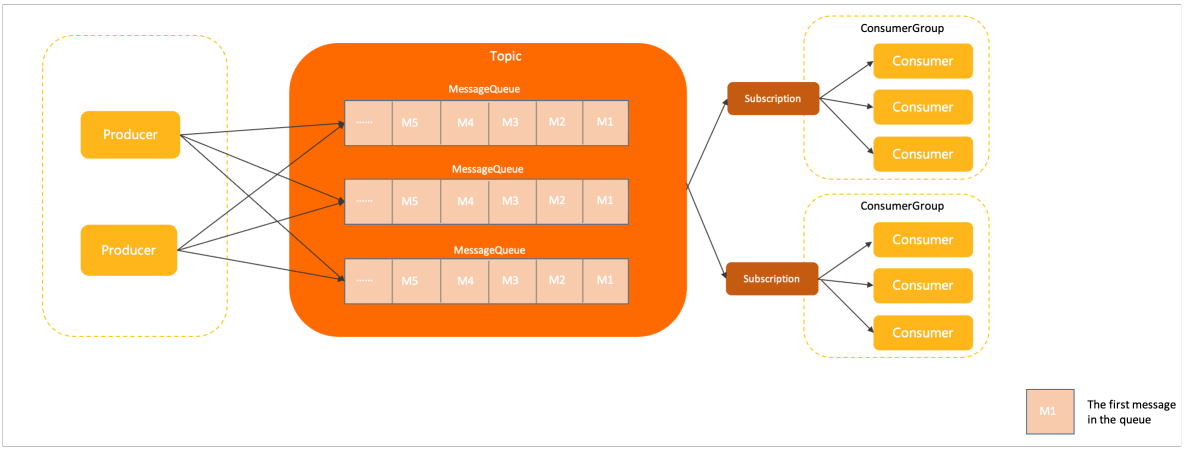

图 1.7: 模型关系：主题

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。

1.3.2　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。

1.3.3 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。

1.3.4 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。

1.4 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。

1.4.1　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）：

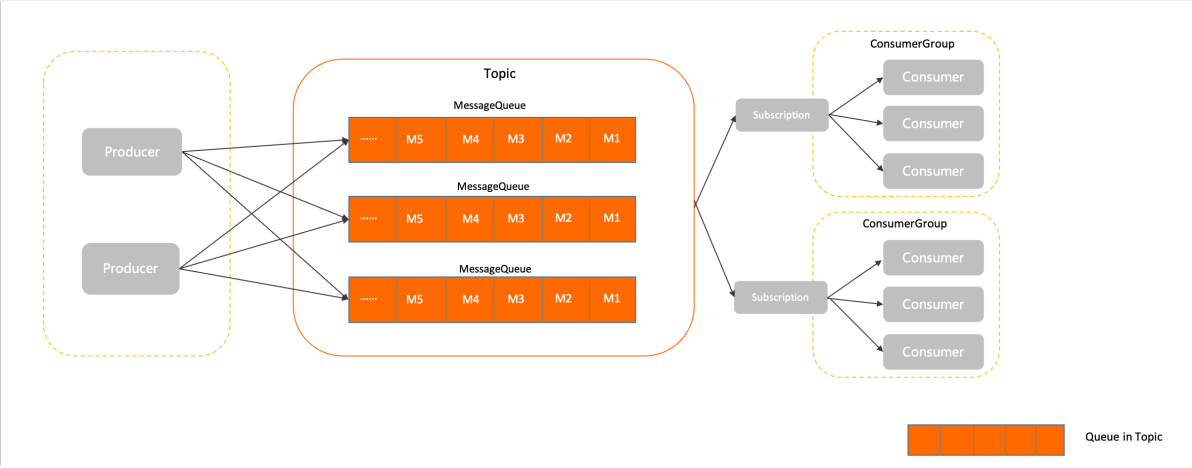

图 1.8: 模型关系：队列

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。

1.4.2　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）。

1.4.3　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。

1.5　消息（Message）

消息是 RocketMQ 中的**最小数据传输单元**。生产者将业务数据的负载和拓展属性包装成消息发送到 RocketMQ 服务端，服务端按照相关语义将消息投递到消费端进行消费。

消息的特点：

1. 不可变性：一旦产生后，消息的内容不会发生改变；消费端获取的消息都是只读消息视图。
2. 持久化：RocketMQ 会默认对消息进行持久化，即将接收到的消息存储到 RocketMQ 服务端的存储文件中，保证消息的可回溯性和系统故障场景下的可恢复性。

1.5.1　模型关系：消息

在整个 RocketMQ 领域模型中，消息所处的位置如下（橙色区域）：

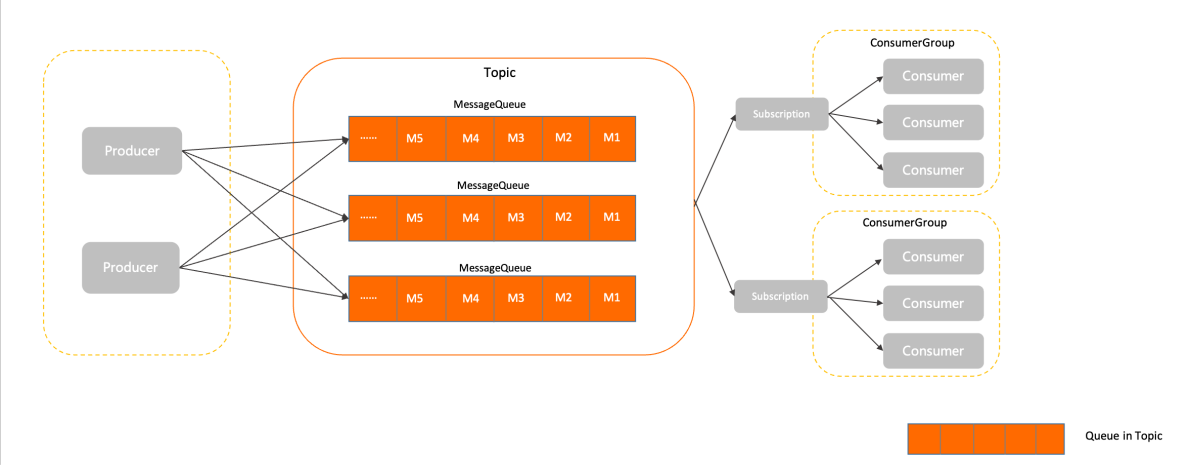

图 1.9: 模型关系：消息

消息由生产者初始化并发送到 RocketMQ 服务端，并按照到达 RocketMQ 服务端的顺序存储到队列中。消费者按照指定的订阅关系从 RocketMQ 服务端中获取消息并消费。

1.5.2　内部属性

1. 主题名称：当前消息所属的主题的名称。
2. 消息队列：实际存储当前消息的队列。
3. 消息位点：当前消息存储在队列中的位置（Offset）。
4. 消息 ID：消息的唯一标识，集群内每条消息的 ID 全局唯一，自动生成的 32 位字符串。
5. 消息类型：当前消息的类型，分为 Normal（普通消息）、FIFO（顺序消息）、Delay（定时消息）、Transaction（事务消息）。
6. 消息负载：业务消息的实际报文数据，由生产者负责序列化编码，按照二进制字节传输。
7. 消息发送时间戳：消息发送时，生产者客户端系统的本地毫秒级时间戳。
8. 消息保存时间戳：消息在 RocketMQ 服务端完成存储时，服务端系统的本地毫秒级时间戳。
9. 消费重试次数：消息消费失败后，RocketMQ 服务端重新投递的次数。每次重试后，重试次数加 1。
10. 其他自定义属性。
11. 索引 Key 列表（可选）：消息的索引键，可通过设置不同的 Key 区分消息和快速查找消息，由生产者定义。
12. 过滤标签 Tag（可选）：消息的过滤标签。消费者可通过 Tag 对消息进行过滤，仅接收指定标签的消息，由生产者定义。

1.5.3　使用建议

- 单条消息不建议传输超大负载。
- 由于消息的不可变性，在使用过程中对消息进行中转操作时，需要对消息重新初始化。

1.6　生产者（Producer）

生产者是 RocketMQ 系统中用来构建并传输消息到 RocketMQ 服务端的运行实体，生产者是匿名的。

在消息生产者中，可以定义如下传输行为：

1. 发送方式：生产者可通过 API 接口设置消息发送的方式，支持同步传输和异步传输。
2. 批量发送：生产者可通过 API 接口设置消息批量传输的方式。例如，批量发送的消息条数或消息大小。
3. 事务行为：Apache RocketMQ 支持事务消息，对于事务消息需要生产者配合进行事务检查

等行为保障事务的最终一致性。

生产者和主题的关系为**多对多关系**，即同一个生产者可以向多个主题发送消息，对于平台类场景如果需要发送消息到多个主题，并不需要创建多个生产者；同一个主题也可以接收多个生产者的消息，以此可以实现生产者性能的水平扩展和容灾。

1.6.1　模型关系：生产者

在整个 RocketMQ 领域模型中，生产者所处的位置如下（橙色区域）：

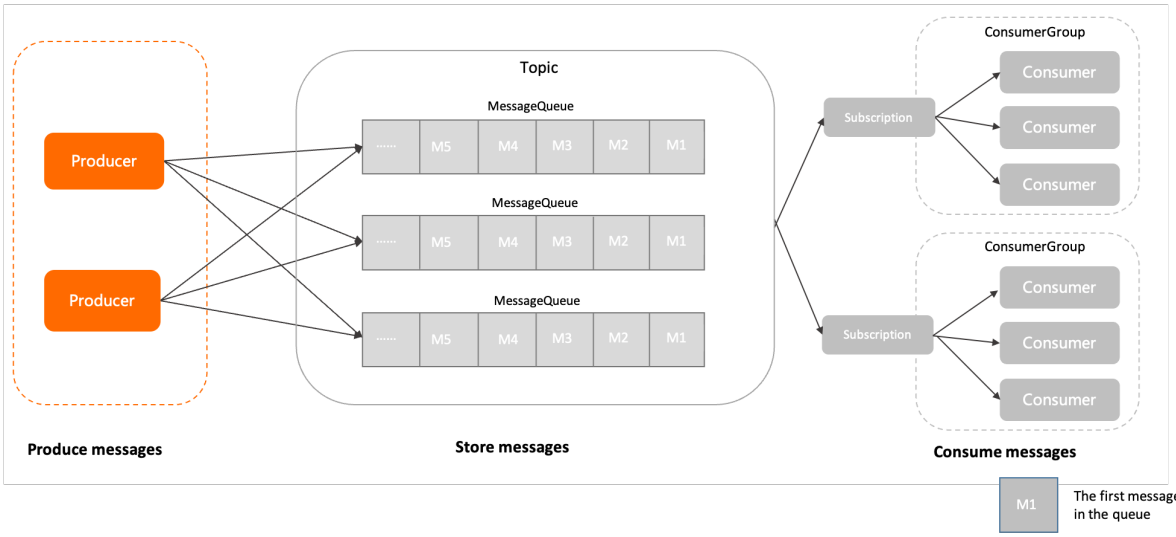

图 1.10: 模型关系：生产者

消息由生产者初始化并发送到 RocketMQ 服务端。

1.6.2　内部属性

1. 生产者客户端 ID：生产者客户端的标识，用于区分不同的生产者。集群内全局唯一。
2. 通信参数：接入点信息（必选，连接服务端的接入地址，用于识别服务端集群）、身份认证信息（可选）、请求超时时间（可选）。
3. 发送重试策略；生产者在消息发送失败时的重试策略。
4. 预绑定主题列表：生产者需要将消息发送到的目标主题列表。服务端会在生产者初始化时根据预绑定主题列表，检查目标主题的访问权限和合法性，而不需要等到应用启动后再检查。事务消息场景下，生产者在故障、重启恢复时，需要检查事务消息的主题中是否有未提交的事务消息，因此必须设置。

5. 事务检查器：事务消息场景下，生产者需要主动实现事务检查器的接口，以保证异常场景下事务的最终一致性。

1.6.3 使用建议

- 对于生产者的创建和初始化，建议遵循够用即可、最大化复用原则，如果有需要发送消息到多个主题的场景，无需为每个主题都创建一个生产者。
- 生产者是可以重复利用的底层资源，类似数据库的连接池中的连接一样，不建议频繁创建和销毁。

1.7 消费者分组（ConsumerGroup）

消费者分组是 RocketMQ 系统中承载多个消费行为一致的消费者的负载均衡分组，并可通过消费者分组内初始化多个消费者实现消费性能的水平扩展以及高可用容灾。和消费者不同，消费者分组并不是运行实体，而是一个**逻辑资源**。

同一分组下的多个消费者将**按照分组内统一的消费行为和负载均衡策略**消费消息，包含以下消费行为：

1. 订阅关系：RocketMQ 以消费者分组的粒度管理订阅关系。
2. 投递顺序性：RocketMQ 的服务端将消息投递给消费者消费时，支持顺序投递和并发投递，投递方式在消费者分组中统一配置。
3. 消费重试策略：消费者消费消息失败时的重试策略，包括重试次数、死信队列设置等。

1.7.1 模型关系：消费者分组

在整个 RocketMQ 领域模型中，消费者分组所处的位置如下（橙色区域）：

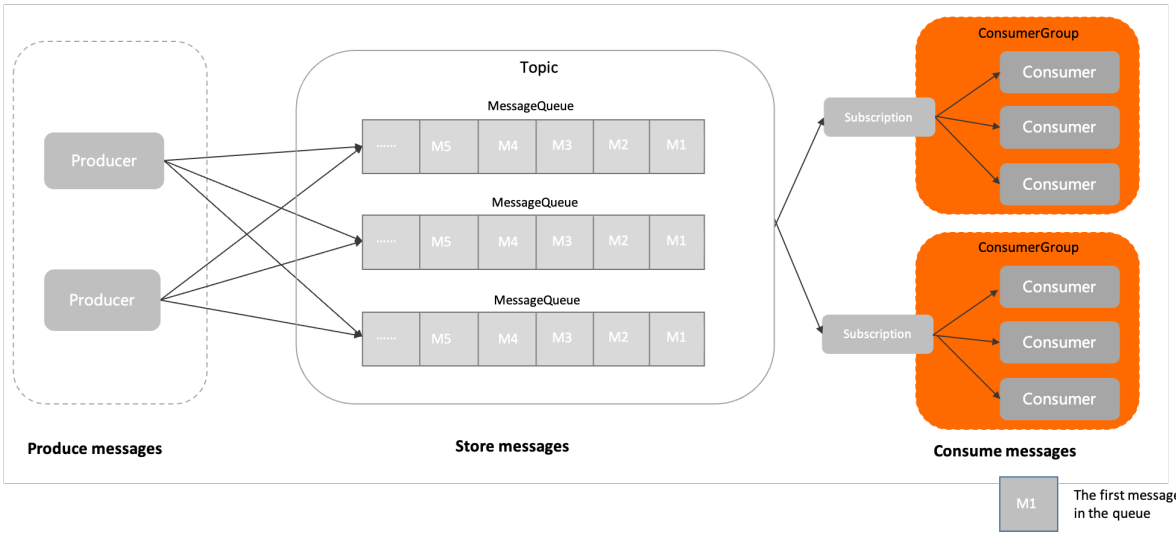

图 1.11: 模型关系：消费者分组

消费者按照指定的订阅关系从 RocketMQ 服务端中获取消息并消费。

1.7.2　内部属性

1. 消费者分组名称：消费者分组的名称，用于区分不同的消费者分组。集群内全局唯一。
2. 订阅关系：当前消费者分组关联的订阅关系集合。包括消费者订阅的主题，以及消息的过滤规则等。订阅关系由消费者动态注册到消费者分组中，RocketMQ 服务端会持久化订阅关系并匹配消息的消费进度。
3. 投递顺序性：消费者消费消息时，RocketMQ 向消费者客户端投递消息的顺序（支持顺序投递和并发投递）。
4. 消费重试策略：消费者消费消息失败时，系统的重试策略（将特定消息投递给消费者重新消费）。包括最大重试次数（超过则放入死信队列）、重试间隔。

1.7.3　使用建议

- 按照业务合理拆分分组。
- 消费者分组管理尽量避免自动化机制，请勿随意进行增、删、改、查操作。

1.8　消费者（Consumer）

消费者是 RocketMQ 中用来接收并处理消息的运行实体。消费者通常被集成在业务系统中，从 RocketMQ 服务端获取消息，并将消息转化成业务可理解的信息，供业务逻辑处理。

在消息消费端，可以定义如下传输行为：

1. 消费者身份：消费者必须关联一个指定的消费者分组，以获取分组内统一定义的行为配置和消费状态。
2. 消费者类型：RocketMQ 面向不同的开发场景提供了多样的消费者类型，包括 PushConsumer 类型、SimpleConsumer 类型、PullConsumer 类型（仅推荐流处理场景使用）等。
3. 消费者本地运行配置：消费者根据不同的消费者类型，控制消费者客户端本地的运行配置。例如消费者客户端的线程数，消费并发度等，实现不同的传输效果。

1.8.1　模型关系：消费者

在整个 RocketMQ 领域模型中，消费者所处的位置如下（橙色区域）：

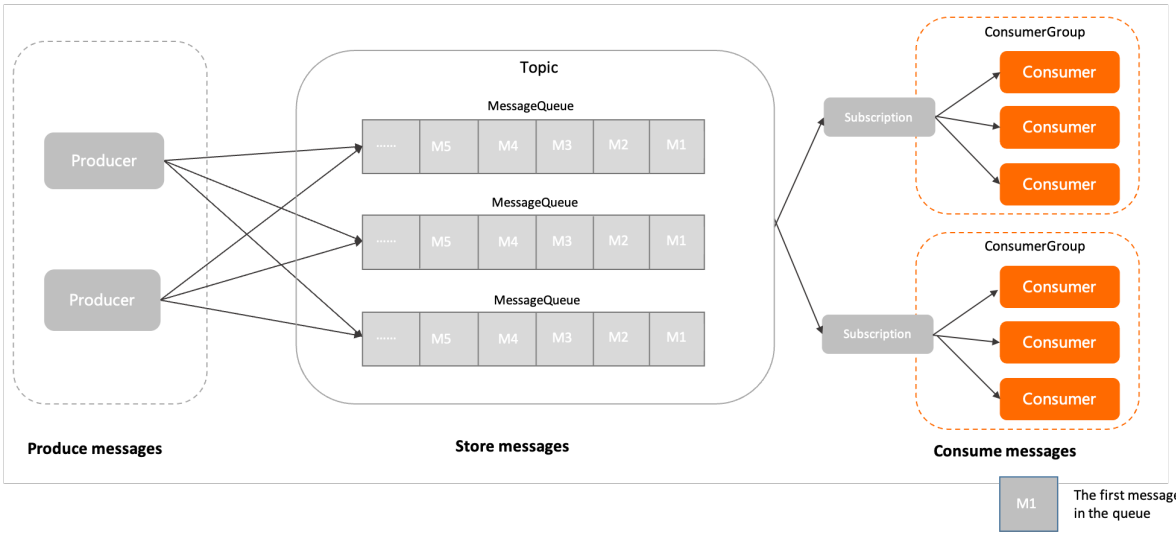

图 1.12: 模型关系：消费者

消费者按照指定的订阅关系从 RocketMQ 服务端中获取消息并消费。

1.8.2　内部属性

1. 客户端 ID：消费者客户端的标识，用于区分不同的消费者。集群内全局唯一。
2. 消费者分组名称：当前消费者关联的消费者分组名称，消费者必须关联到指定的消费者分组，通过消费者分组获取消费行为。
3. 预绑定订阅关系列表：指定消费者的订阅关系列表。RocketMQ 服务端可在消费者初始化阶段，根据预绑定的订阅关系列表对目标主题进行权限及合法性校验，无需等到应用启动后才能校验。

4. 通信参数：接入点信息（必选）、身份认证信息（可选）、请求超时时间（可选）。
5. 消费监听器：获得 RocketMQ 服务端的消息后，消费者调用消息消费逻辑的监听器。

1.8.3 使用建议

- 消费者是可以重复利用的底层资源，类似数据库的连接池中的连接一样，不建议频繁创建和销毁。
- 由于消费者在通信协议层面支持非阻塞传输模式，大部分场景下，单台主机内的一个消费分组只需要初始化一个唯一的消费者即可。

1.9 订阅关系（Subscription）

订阅关系是 RocketMQ 系统中消费者获取消息、处理消息的规则和状态配置。订阅关系由消费者分组动态注册到服务端系统，并在后续的消息传输中按照订阅关系定义的过滤规则进行消息匹配和消费进度维护。

通过配置订阅关系，可控制如下传输行为：

1. 消息过滤规则：用于控制消费者在消费消息时，选择主题内的哪些消息进行消费，设置消费过滤规则可以高效地过滤消费者需要的消息集合，灵活根据不同的业务场景设置不同的消息接收范围。
2. 消费状态：Apache RocketMQ 服务端默认提供订阅关系持久化的能力，即消费者分组在服务端注册订阅关系后，当消费者离线并再次上线后，可以获取离线前的消费进度并继续消费。

1.9.1 模型关系：订阅关系

在整个 RocketMQ 领域模型中，订阅关系所处的位置如下（橙色区域）：

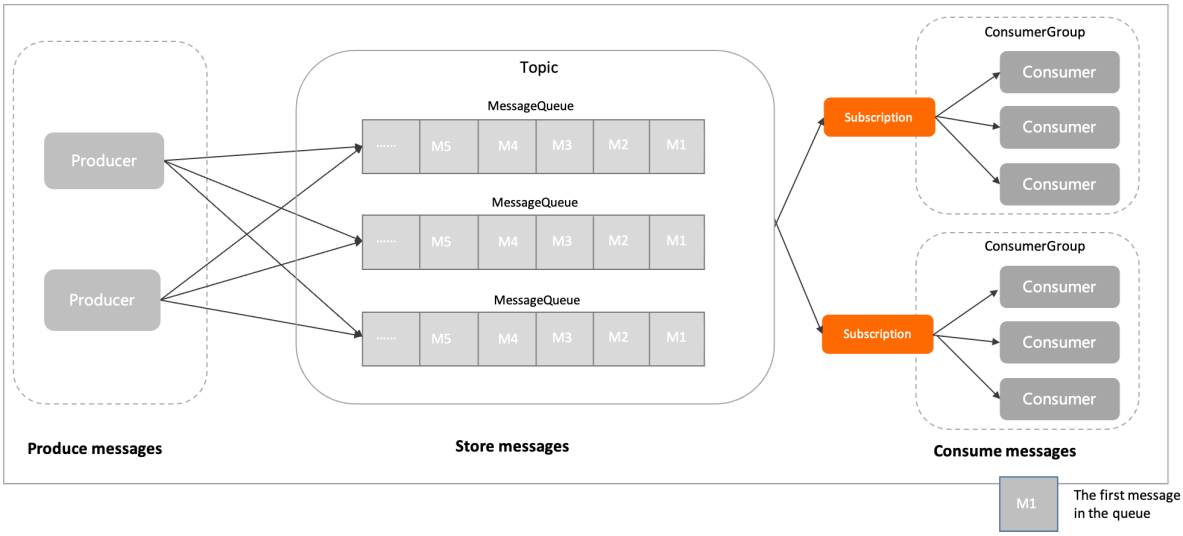

图 1.13: 模型关系：订阅关系

消费者按照指定的订阅关系从 RocketMQ 服务端中获取消息并消费。

1.9.2　订阅关系判断原则

RocketMQ 的订阅关系**按照消费者分组和主题粒度**设计，因此，一个订阅关系指的是指定某个消费者分组对于某个主题的订阅。

判断原则如下：

不同消费者分组对于同一个主题的订阅

不同消费者分组对于同一个主题的订阅是相互独立的。如图：

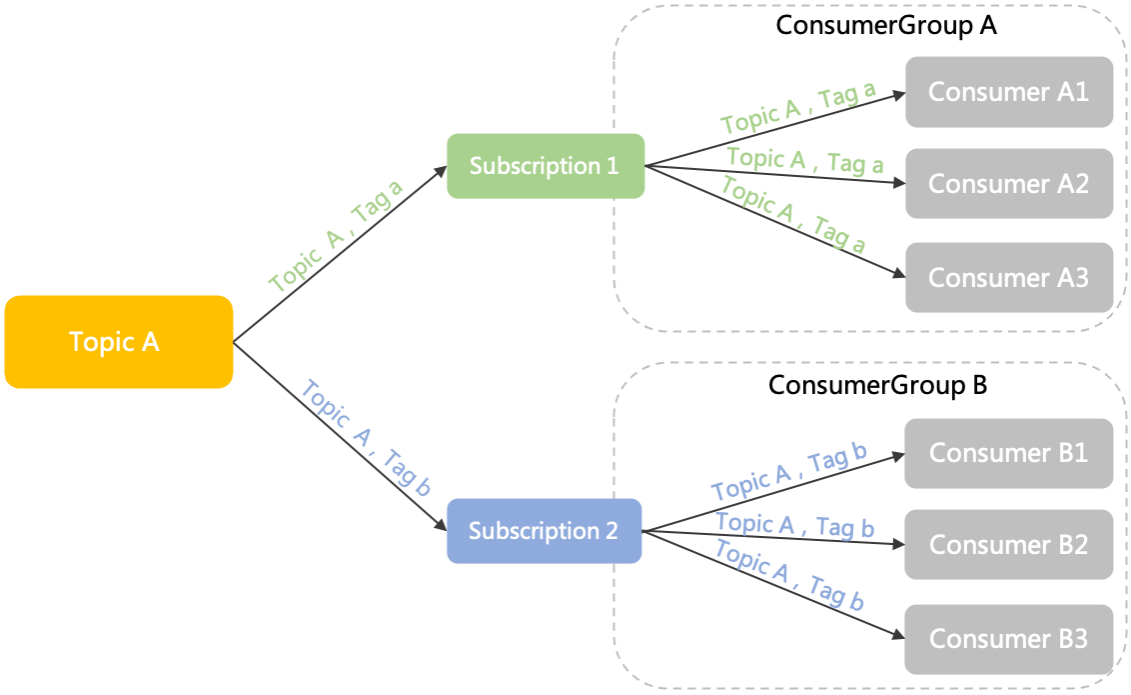

图 1.14: 不同消费者分组对于同一个主题的订阅

消费者分组 Group A 和消费者分组 Group B 分别以不同的订阅关系订阅了同一个主题 Topic A，这两个订阅关系互相独立，可以各自定义，不受影响。

同一个消费者分组对于不同主题的订阅

同一个消费者分组对于不同主题的订阅也是相互独立的。如图：

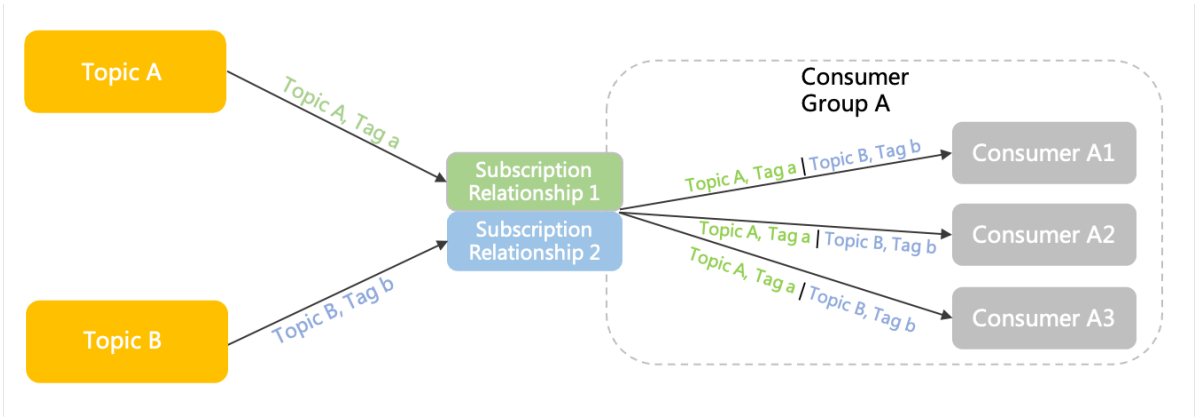

图 1.15: 同一个消费者分组对于不同主题的订阅

消费者分组 Group A 订阅了两个主题 Topic A 和 Topic B，对于 Group A 中的消费者来说，

订阅的 Topic A 为一个订阅关系，订阅的 Topic B 为另外一个订阅关系，且这两个订阅关系互相独立，可以各自定义，不受影响。

1.9.3 内部属性

1. 过滤类型：消息过滤规则的类型，分为 TAG 过滤（对 Tag 字符串进行全文过滤匹配）和 SQL92 过滤（按照 SQL 语法对消息属性进行过滤匹配）。
2. 过滤表达式：自定义的过滤规则表达式。

1.9.4 使用建议

- 不要频繁修改订阅关系，这样可能会导致客户端一直处于负载均衡调整和变更的过程，从而影响消息接收。