RocketMQ 消息集成：多类型业务消息——定时消息

引言

Apache RocketMQ 诞生至今，历经十余年大规模业务稳定性打磨，服务了 100% 阿里集团内部业务以及阿里云数以万计的企业客户。作为金融级可靠的业务消息方案，RocketMQ 从创建之初就一直专注于业务集成领域的异步通信能力构建。

本篇将继续业务消息集成的场景，从使用场景、应用案例、功能原理以及最佳实践等角度介绍 RocketMQ 的定时消息功能。

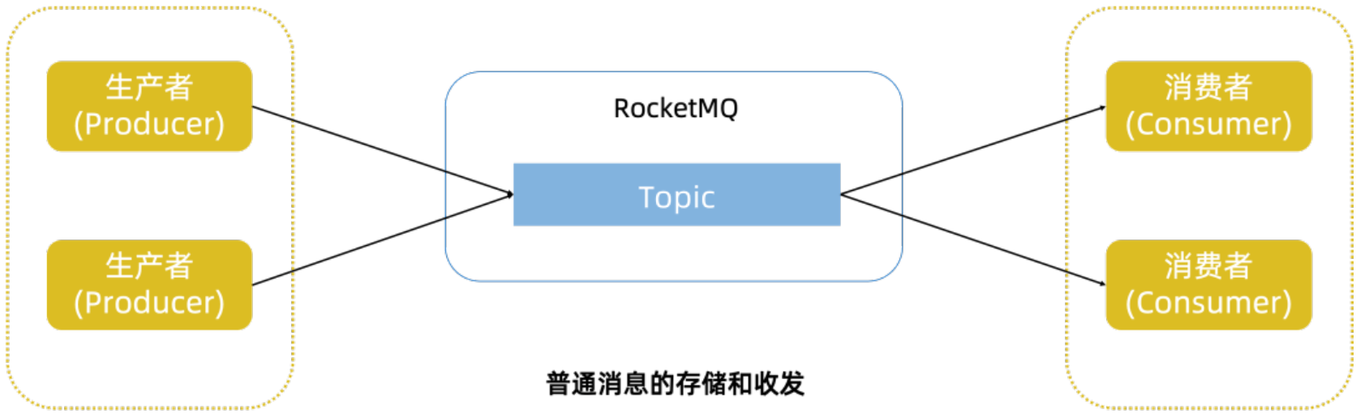
点击下方链接，查看直播讲解： https://yqh.aliyun.com/live/detail/29063

概念：什么是定时消息

在业务消息集成场景中，定时消息是，生产者将一条消息发送到消息队列后并不期望这条消息马上会被消费者消费到，而是期望到了指定的时间，消费者才可以消费到。

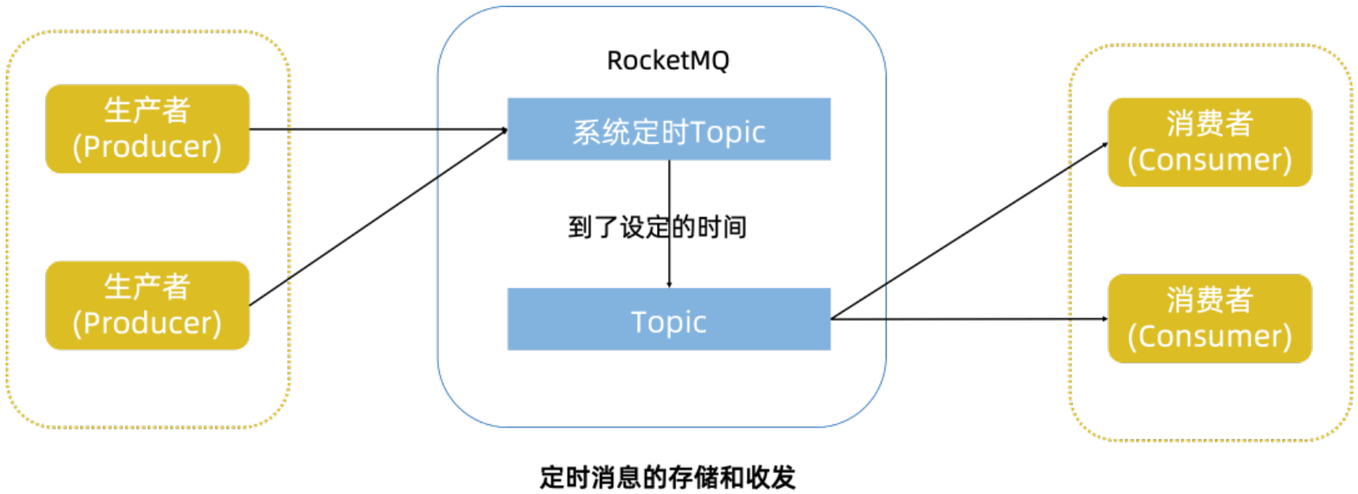
相似地，延迟消息其实是对于定时消息的另外一种解释，指的是生产者期望消息延迟一定时间，消费者才可以消费到。可以理解为定时到当前时间加上一定的延迟时间。

对比一下定时消息和普通消息的流程。普通消息，可以粗略的分为消息发送，消息存储和消息消费三个过程。当一条消息发送到 Topic 之后，那么这条消息就可以马上处于等待消费者消费的状态了。



普通消息的存储和收发

而对于定时/延时消息来说，其可以理解为在普通消息的基础上叠加了定时投递到消费者的特性。生产者发送了一条定时消息之后，消息并不会马上进入用户真正的Topic里面，而是会被 RocketMQ 暂存到一个系统 Topic 里面，当到了设定的时间之后，RocketMQ 才会将这条消息投递到真正的 Topic 里面，让消费者可以消费到。



定时消息的存储和收发

场景：为什么需要使用定时消息

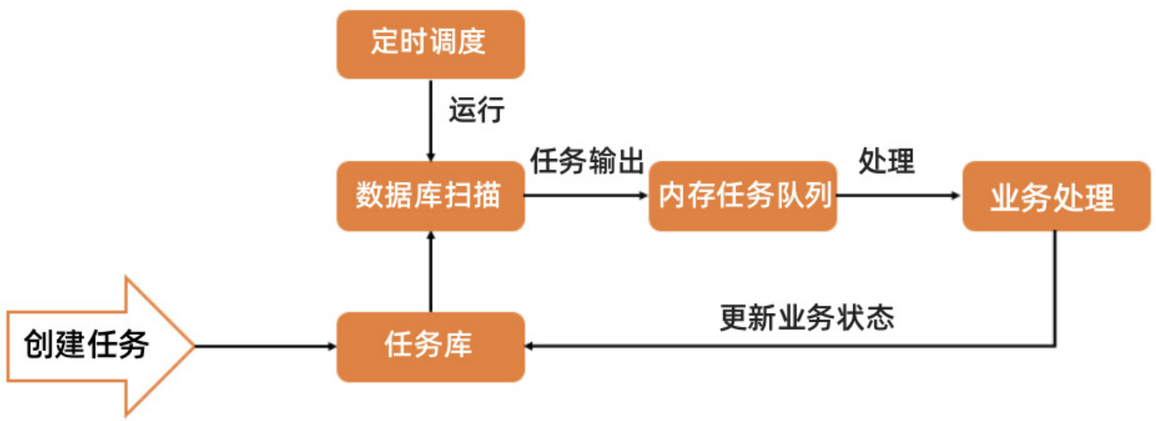
在分布式定时调度触发、任务超时处理等场景，需要实现精准、可靠的定时事件触发。往往这类定时事件触发都会存在以下诉求：

- 高性能吞吐：需要大量事件触发，不能有性能瓶颈。
- 高可靠可重试：不能丢失事件触发。
- 分布式可扩展：定时调度不能是单机系统，需要能够均衡的调度到多个服务负载。

传统的定时调度方案，往往基于数据库的任务表扫描机制来实现。大概的思路就是将需要定时触发的任务放到数据库，然后微服务应用定时触发扫描数据库的操作，实现任务捞取处理。

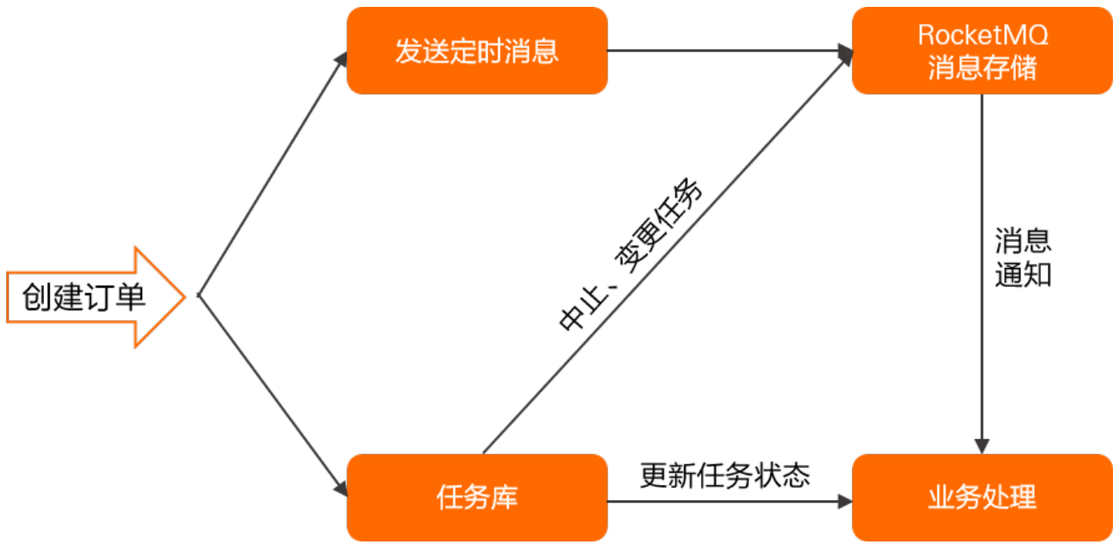
这类方案虽然可以实现定时调度，但往往存在很多不足之处：

- 重复扫描：在分布式微服务架构下，每个微服务节点都需要去扫描数据库，带来大量冗余的任务处理，需要做去重处理。
- 定时间隔不准确：基于定时扫描的机制无法实现任意时间精度的延时调度。
- 横向扩展性差：为规避重复扫描的问题，数据库扫表的方案里往往会按照服务节点拆分表，但每个数据表只能被单节点处理，这样会产生性能瓶颈。



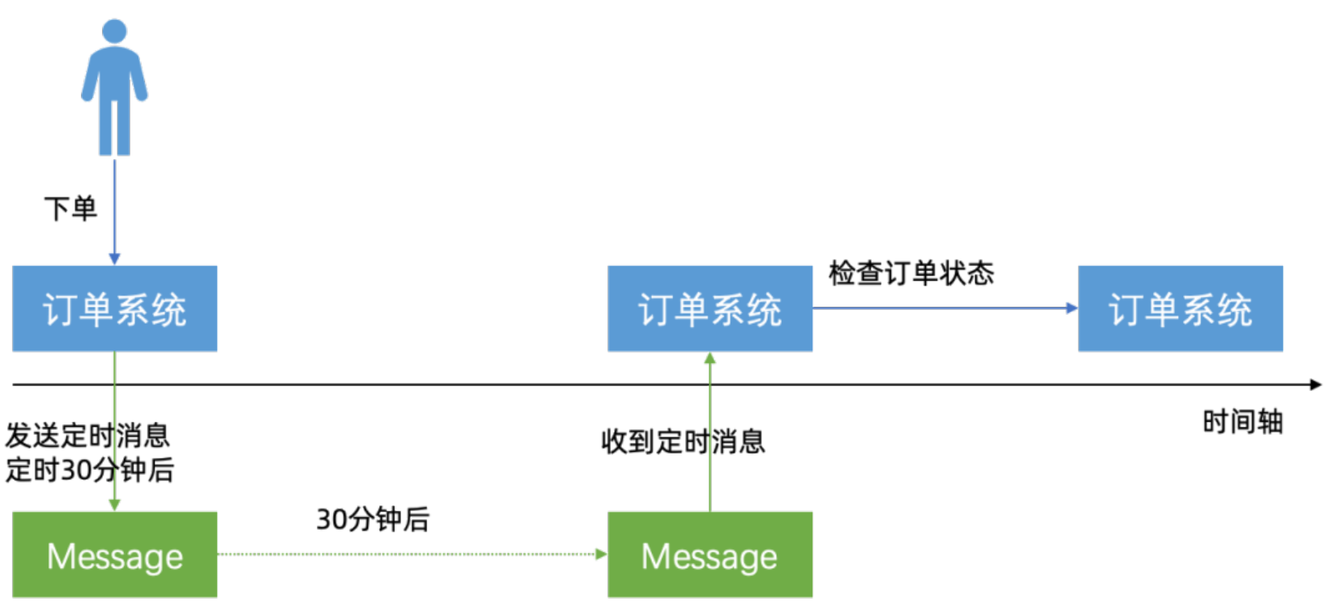
在这类定时调度类场景中，使用 RocketMQ 的定时消息可以简化定时调度任务的开发逻辑，实现高性能、可扩展、高可靠的定时触发能力。

- 精度高、开发门槛低：基于消息通知方式不存在定时阶梯间隔。可以轻松实现任意精度事件触发，无需业务去重。
- 高性能可扩展：传统的数据库扫描方式较为复杂，需要频繁调用接口扫描，容易产生性能瓶颈。消息队列 RocketMQ 版的定时消息具有高并发和水平扩展的能力。



案例：使用定时消息实现金融支付超时需求

利用定时消息可以实现在一定的时间之后才进行某些操作而业务系统不用管理定时的状态。下面介绍一个典型的案例场景：金融支付超时。现在有一个订单系统，希望在用户下单 30 分钟后检查用户的订单状态，如果用户还没有支付，那么就自动取消这笔订单。

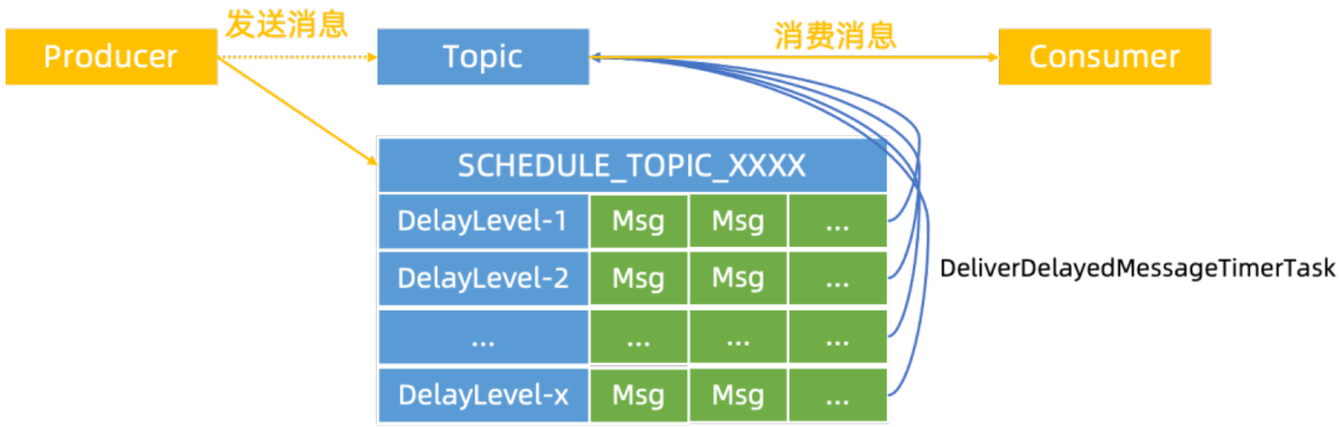


基于 RocketMQ 定时消息，我们可以在用户下单之后发送一条定时到 30 分钟之后的定时消息。同时，我们可以使用将订单 ID 设置为 MessageKey。当 30 分钟之后，订单系统收到消息之后，就可以通过订单 ID 检查订单的状态。如果用户超时未支付，那么就自动的将这笔订单关闭。

原理：RocketMQ 定时消息如何实现

固定间隔定时消息

如前文介绍，定时消息的核心是如何在特定的时间把处于系统定时 Topic 里面的消息转移到用户的 Topic 里面去。

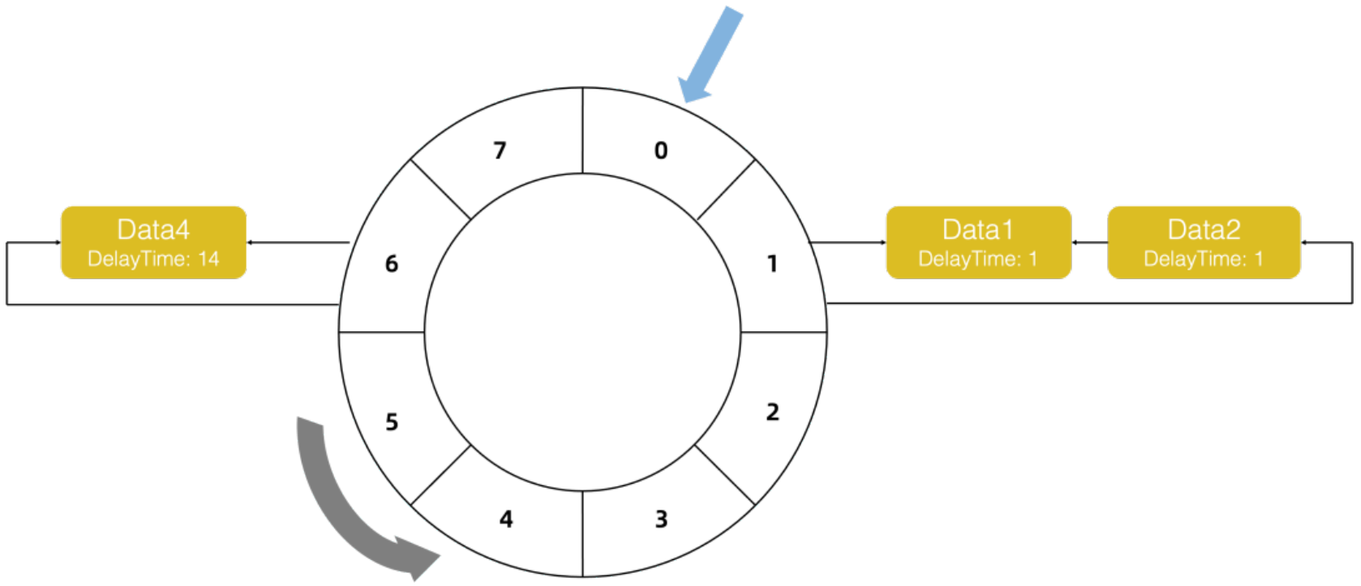


Apache RocketMQ 4.x 的版本的定时消息是先将定时消息放到按照 DelayLevel 放到 SCHEDULE_TOPIC_XXXX 这个系统的不同 Queue 里面，然后为每一个 Queue 启动一个定时任务，定时的拉取消息并将到了时间的消息转投到用户的 Topic 里面去。这样虽然实现简单，但也导致只能支持特定 DelayLevel 的定时消息。

当下，支持定时到任意秒级时间的定时消息的实现的 pr 提出到了社区，下面简单的介绍一下其基本的实现原理。

时间轮算法

在介绍具体的实现原理之前，先介绍一下经典的时间轮算法，这是定时消息实现的核心算法。



如上所示，这是一个一圈定时为 7 秒的时间轮，定时的最小精度的为秒。同时，时间轮上面会有一个指向当前时间的指针，其会定时的移向下一个刻度。

现在我们想定时到 1 秒以后，那么就将数据放到 “1” 这个刻度里面，同时如果有多个数据需要定时到同一个时间，

那么会以链表的方式添加到后面。当时间轮转到 “1” 这个刻度之后，就会将其读取并从链表出队。那如果想定到超过时间轮一圈的时间怎么处理呢？例如我们想定时到 14 秒，由于一圈的时间是 7 秒，那么我们将其放在“6”这个刻度里面。当第一次时间轮转到“6” 时，发现当前时间小于期望的时间，那么忽略这条数据。当第二次时间轮转到“6”时，这个时候就会发现已经到了我们期望的 14 秒了。

任意秒级定时消息

在 RocketMQ 中，使用 TimerWheel 对于时间轮进行描述和存储，同时使用一个 AppendOnly 的 TimerLog 记录时间轮上面每一个刻度所对应的所有的消息。

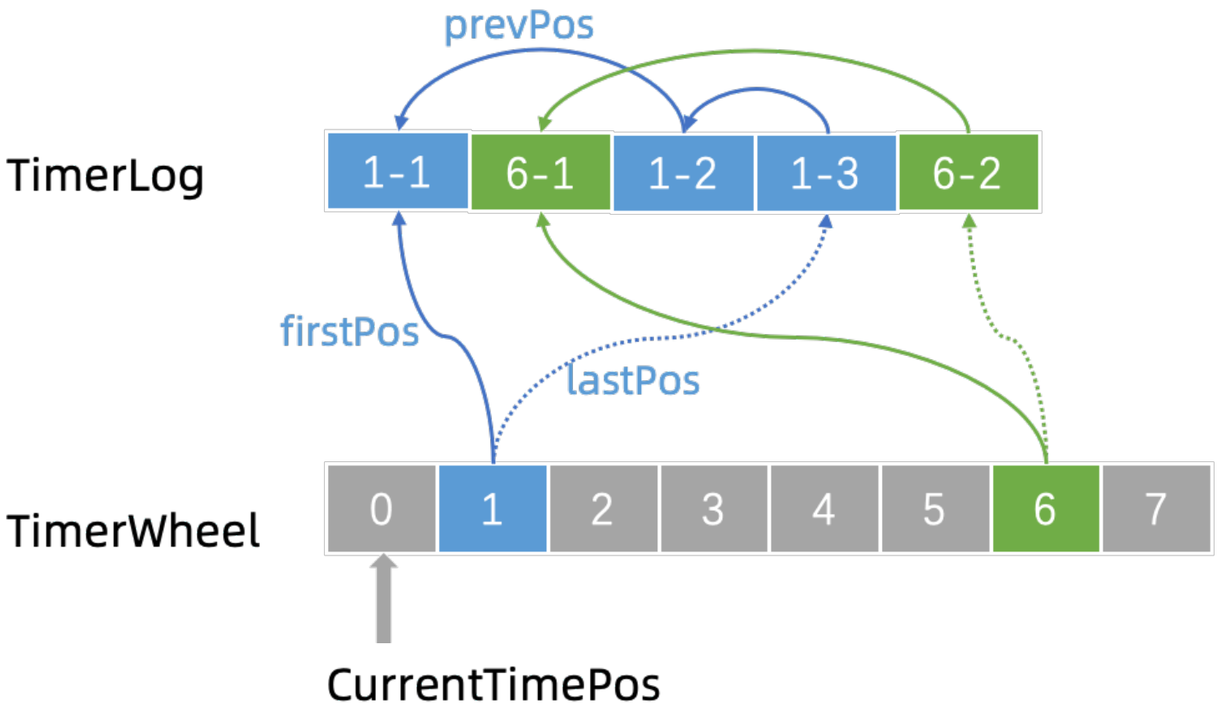
TimerLog 记录了一条定时消息的一些重要的元数据，用于后面定时的时间到了之后，将消息转移到用户的 Topic 里面去。其中几个重要的属性如下：

名称	描述
size	一个条记录的大小
prevPos	指向前一条记录
magic	magic value
delayTime	定时的时间
offsetReal	该条消息在CommitLog中的offset
sizeReal	该条消息在CommitLog中的大小

对于 TimerWheel 来说，可以抽象的认为是一个定长的数组，数组中的每一格代表时间轮上面的一个“刻度”。TimerWheel 的一个“刻度”拥有以下属性。

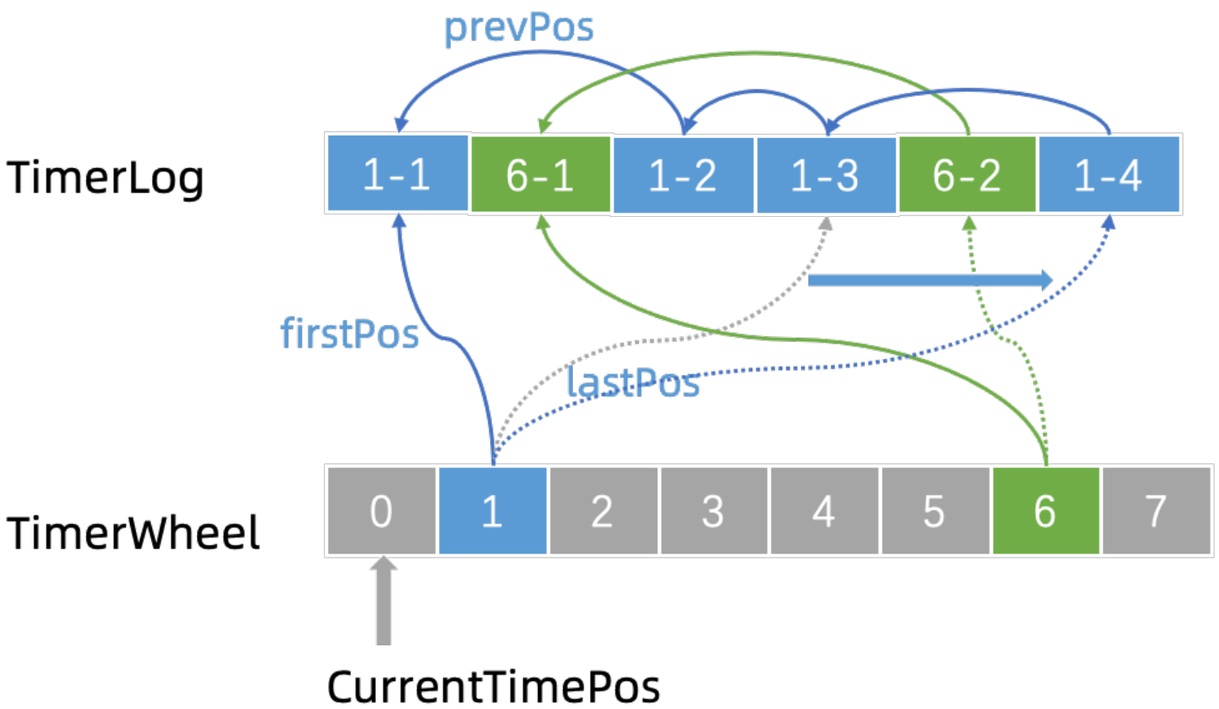
名称	描述
delayTime	延迟时间
firstPos	首条TimerLog的记录的位置
lastPos	最后一条TimerLog的记录的位置

TimerWheel 和 TimerLog 直接的关系如下图所示：



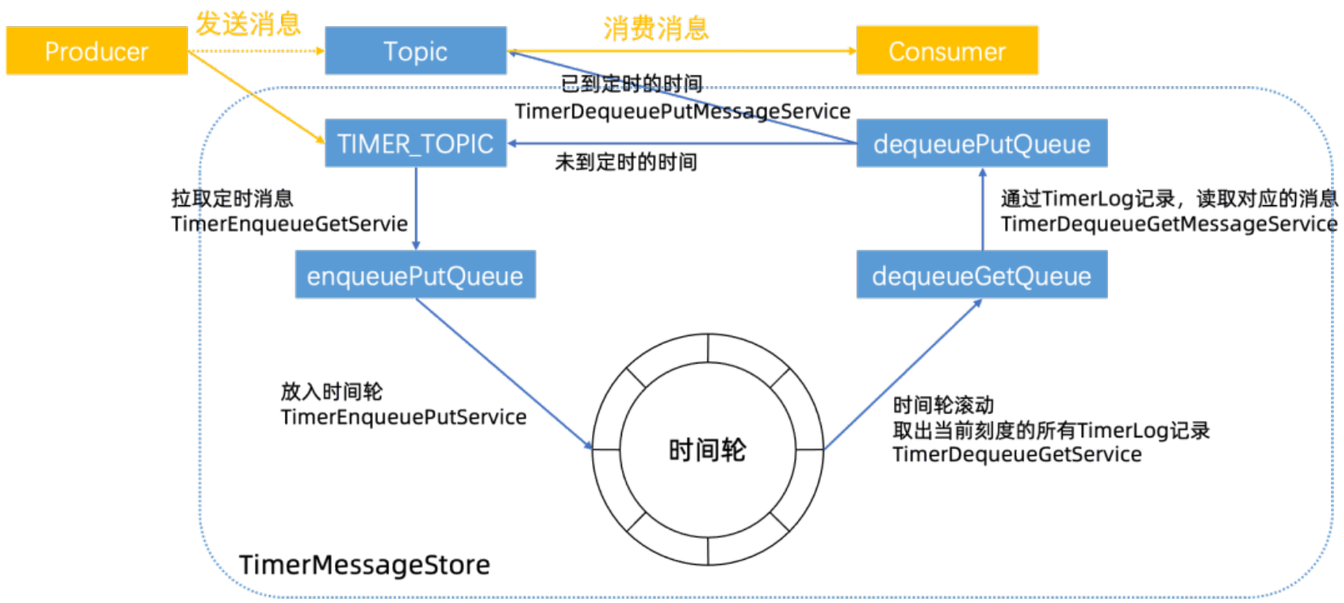
TimerWheel 中的每一格代表着一个时间刻度，同时会有一个 firstPos 指向这个刻度下所有定时消息的首条 TimerLog 记录的地址，一个 lastPos 指向这个刻度下所有定时消息最后一条 TimerLog 的记录的地址。并且，对于所处于同一个刻度的的消息，其 TimerLog 会通过

prevPos 串联成一个链表。



当需要新增一条记录的时候，例如现在我们要新增一个 “1-4”。那么就将新记录的 prevPos 指向当前的 lastPos，即 “1-3”，然后修改 lastPos 指向 “1-4”。这样就将同一个刻度上面的 TimerLog 记录全都串起来了。

有了 TimerWheel 和 TimerLog 之后，我们再来看一下一条定时消息从发送到 RocketMQ 之后是怎么最终投递给用户的。



首先，当发现用户发送的是一个定时消息过后，RocketMQ 实际上会将这条消息发送到一个专门用于处理定时消息的系统 Topic 里面去

然后在 TimerMessageStore 中会有五个 Service 进行分工合作，但整体可以分为两个阶段：入时间轮和出时间轮

对于入时间轮：

- TimerEnqueueGetService 负责从系统定时 Topic 里面拉取消息放入 enqueuePutQueue 等待 TimerEnqueuePutService 的处理
- TimerEnqueuePutService 负责构建 TimerLog 记录，并将其放入时间轮的对应的刻度中

对于出时间轮：

- TimerDequeueGetService 负责转动时间轮，并取出当前时间刻度的所有 TimerLog 记录放入 dequeueGetQueue
- TimerDequeueGetMessageService 负责根据 TimerLog 记录，从 CommitLog 中读取消息
- TimerDequeuePutMessageService 负责判断队列中的消息是否已经到期，如果已经到期了，那么将其投入用户的 Topic 中，等待消费消费；如果还没有到期，那么重新投入系统定时 Topic，等待重新进入时间轮。

实战：使用定时消息

了解了 RocketMQ 秒级定时消息的原理后，我们看下如何使用定时消息。首先，我们需要创建一个“定时/延时消息”类型的 Topic，可以使用控制台或者 CLi 命令创建。

从前面可以看出，对于定时消息来说，是在发送消息的时候“做文章”。所以，对于生产者，相对于发送普通消息，我们可以在发送的时候设置期望的投递时间。

Java 复制代码

```java
public static void main(String[] args) {
    ClientServiceProvider provider = ClientServiceProvider.loadService();
    StaticSessionCredentialsProvider staticSessionCredentialsProvider =
        new StaticSessionCredentialsProvider("AccessKey", "SecretKey");
    ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
        .setEndpoints("Endpoint")
        .setCredentialProvider(staticSessionCredentialsProvider)
        .build();

    try {
        Producer producer = provider.newProducerBuilder()
            .setClientConfiguration(clientConfiguration)
            .build();
        Message delayMessage = provider.newMessageBuilder()
            .setTopic("Topic")
            .setKeys("Key")
            .setTag("Tag")
            .setBody("MessageBody".getBytes())
            // 设置定时到 5秒之后
            .setDeliveryTimestamp(System.currentTimeMillis() +
    Duration.ofSeconds(5).toMillis())
                .build();
            SendReceipt sendReceipt = producer.send(delayMessage);
            // 发送消息成功后，记录 MessageID可用于异常问题的排查
            System.out.println(sendReceipt.getMessageId());
        } catch (ClientException e) {
            // 捕获异常，进行异常处理
        }
    }
```

当定时的时间到了之后，这条消息其实就是一条投递到用户 Topic 的普通消息而已。所以对于消费者来说，和普通消息的消费没有区别。

Java 复制代码

```java
public static void main(String[] args) {
    ClientServiceProvider provider = ClientServiceProvider.loadService();
    StaticSessionCredentialsProvider staticSessionCredentialsProvider =
        new StaticSessionCredentialsProvider("AccessKey", "SecretKey");
    ClientConfiguration clientConfiguration = ClientConfiguration.newBuilder()
        .setEndpoints("Endpoint")
        .setCredentialProvider(staticSessionCredentialsProvider)
        .build();

    try {
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            .setConsumerGroup("ConsumerGroup")
            .setSubscriptionExpressions(Collections.singletonMap("Topic", new
FilterExpression()))
            .setMessageListener(messageView -> {
                System.out.println("Receive Message " +
messageView.getMessageId());
                return ConsumeResult.SUCCESS;
            })
            .build();
    } catch (ClientException e) {
        // 捕获异常，进行异常处理
    }
}
```

注意：定时消息的实现逻辑需要先经过定时存储等待触发，定时时间到达后才会被投递给消费者。因此，如果将大量定时消息的定时时间设置为同一时刻，则到达该时刻后会有大量消息同时需要被处理，会造成系统压力过大。所以一般建议尽量不要设置大量相同触发时刻的消息。

活动推荐

阿里云基于 Apache RocketMQ 构建的企业级产品-消息队列RocketMQ 5.0版现开启活动：

1、新用户首次购买包年包月，即可享受全系列 85折优惠！ 了解活动详情：https://www.aliyun.com/product/rocketmq

更多内容：

欢迎钉钉扫描二维码加入
「Apache RocketMQ 中国开发者钉钉群」

欢迎扫描二维码
关注Apache RocketMQ公众号