18.图解分布式消息中间件在分布式环境下并发流量控制的应用

大家好，我是隐墨星辰，深耕境内/跨境支付架构设计十余年。

本篇重点讲清楚分布式消息中间件的特点，常见消息中间件的简单对比，在支付系统的应用场景，比如削峰填谷，系统应用间的解耦，事务消息等。

内容偏入门介绍，已经使用过消息中间件的同学可以不用往下看了。

1. 前言

在流量控制系列文章中的前四篇，分别介绍了固定时间窗口算法、滑动时间窗口算法、漏桶原理、令牌桶原理，应用场景和java版本的核心代码。

我们做个简单回顾：

固定窗口：算法简单，对突然流量响应不够灵活。超过流量的会直接拒绝，通常用于限流。

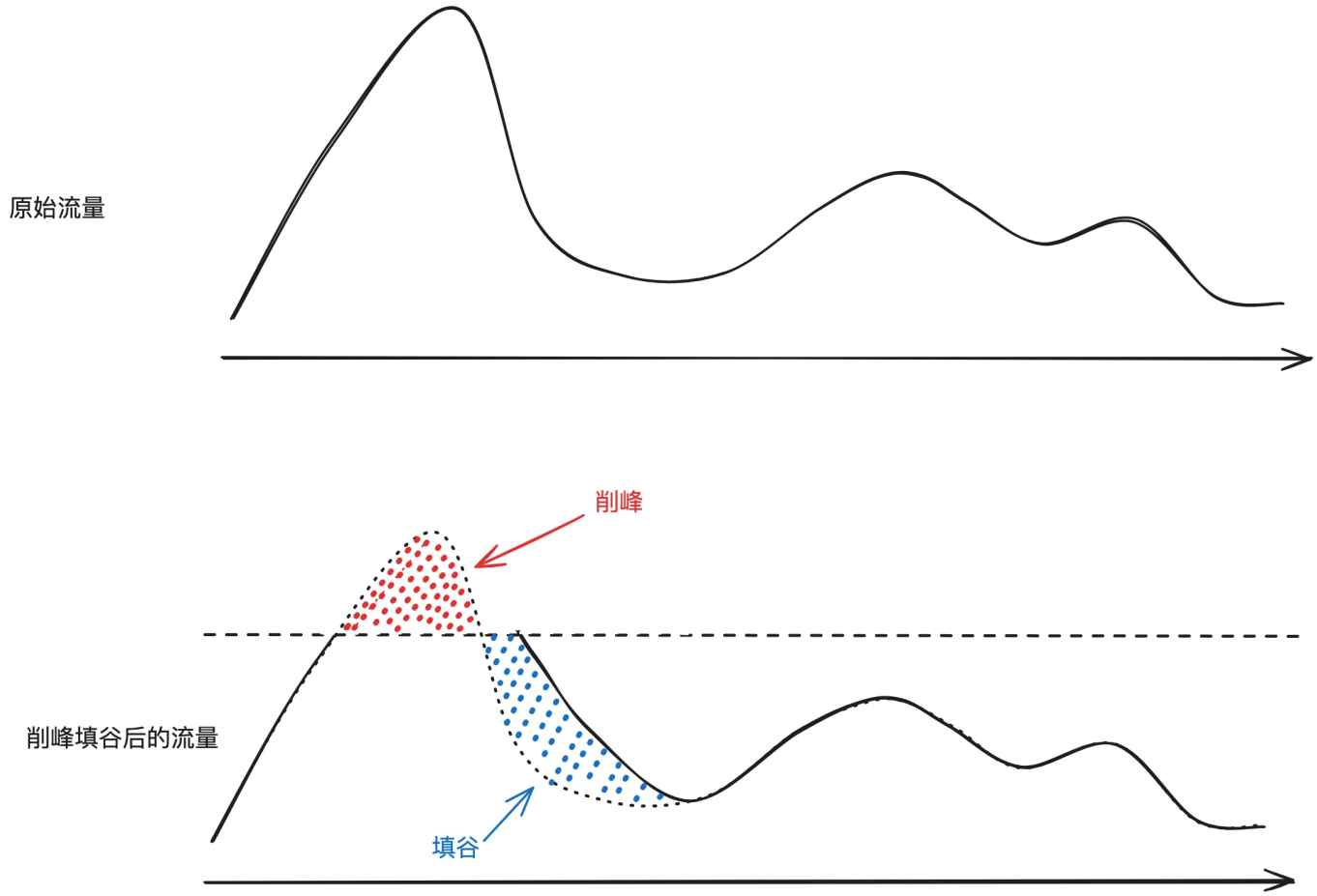
滑动窗口： 算法简单，对突然流量响应比固定窗口灵活。超过流量的会直接拒绝，通常用于限流。

漏桶算法：在固定窗口的基础之上，使用队列缓冲流量。提供了稳定的流量输出，适用于对流量平滑性有严格要求的场景。

令牌桶算法：在滑动窗口的基础之上，使用队列缓冲流量。提供了稳定的流量输出，且能应对突发流量。

今天讲的分布式消息中间件在支付场景的削峰填谷和应用间解耦用得比较多，且对精度没有那么苛刻的场景，比如集群低到1TPS，就无法做到。

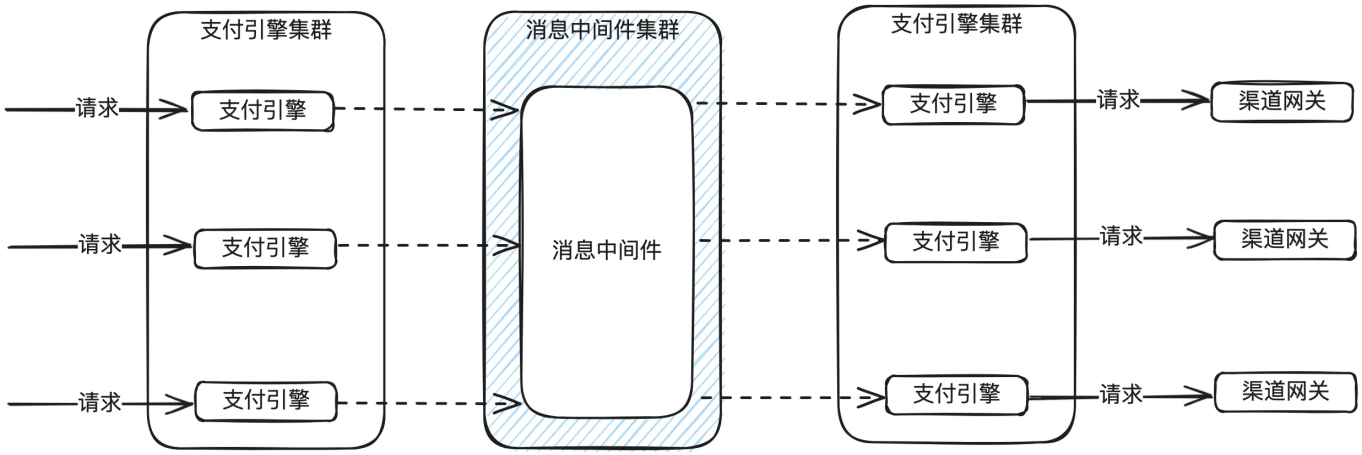
2. 削峰填谷原理



削峰：在流量高峰期，通过消息中间件暂存大量的请求，减少对后端系统的直接压力。

填谷：在低峰期，逐渐处理这些积累的请求。

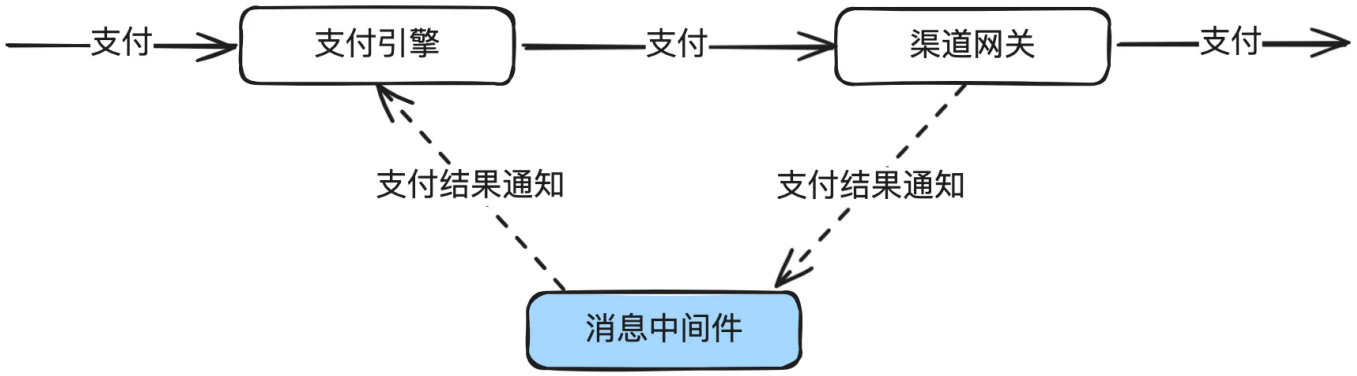
这种方法能有效平衡系统负载，防止在高峰时段系统崩溃。



我们一般使用消息中间件实现削峰填谷。下面是一个支付引擎自产生消的示例图。收到支付请求后，先扔到消息中间件，然后启用新的监听线程去消费。通过控制消费线程数来控制流量。

比如一下来了1000个请求，一共10台机器，每台机器消费线程只开5个，每个请求处理500ms，那么每秒就处理100个请求，共耗时10秒处理完。

消息中间件还有一个作用，就是**应用间的解耦**。比如支付成功后，渠道网关通过消息中间件返回给支付引擎。



3. 常见的分布式消息中间件介绍

消息中间件有很多，这里简单对比 RocketMQ、RabbitMQ 和 Kafka在性能、可靠性、易用性、功能特性、适用场景等方面的不同。如下：

性能

- RocketMQ: 提供非常高的性能和吞吐量，特别适合大规模的消息传输和处理场景。
- RabbitMQ: 性能优秀，尤其在小型消息的传递上非常高效。但在处理非常大量的消息时，性

能可能不如 Kafka 和 RocketMQ。

- Kafka: 专为高吞吐量设计，特别适合需要处理大数据流的场景。它在持久化和分布式处理方面的性能表现尤其出色。

可靠性

- RocketMQ: 提供高可靠性保证，支持分布式事务。
- RabbitMQ: 通过消息持久化、交付确认等机制提供可靠的消息服务。
- Kafka: 数据持久化和高容错能力，确保了高可靠性。

易用性和管理

- RocketMQ: 相对复杂，需要一定的学习曲线，但提供丰富的特性和灵活性。
- RabbitMQ: 用户友好，易于安装和配置，拥有直观的管理界面。
- Kafka: 配置和管理相对复杂，但社区支持强大，提供了丰富的文档资源。

功能特性

- RocketMQ: 支持广泛的消息模式，包括顺序消息、定时/延时消息和事务消息。
- RabbitMQ: 提供多种消息路由模式，支持灵活的消息模型和多种协议。
- Kafka: 专注于高吞吐量的消息队列和流处理，支持实时数据处理。

适用场景

- RocketMQ: 适用于大规模分布式系统中的消息处理，如电商平台和金融系统。
- RabbitMQ: 适合需要复杂路由、多种消息协议和高效小消息处理的场景，如企业应用集成。
- Kafka: 非常适合于需要高吞吐量、大数据处理和实时流处理的应用，如日志聚合和实时监控系统。

结论

- 在支付系统，我个人更推荐RocketMQ，原因有两个：第一，经过阿里电商+支付各种非常高并发的大促洗礼，RocketMQ 在大型分布式系统中表现出色。第二，支持事务消息，这个在支付领域还是很有用的。

另外，我也只是简单介绍，大家实际应用的时候，根据实际情况去选型，建议是做多方了解后，部署验证后才规模使用。不过话说回来，这几款分布式消息中间件的技术非常成熟了，除了几个顶流的互联网公司外，基本可以随便用，哪个手熟就使用哪个。

4. 使用注意事项

脑裂问题

曾经在生产环境碰到过RabbitMQ脑裂问题，交易全部中断。在分布式环境中完全避免也不现实，建议加强监控。

消费线程数问题

消费线程要合理设置，太多，可能达不到削峰填谷的效果，太少，消息有可能会积累，影响处理时效。

比如支付是有时效，积压太久就会导致用户放弃支付。

消息积压应对

提前做好预估，以及监控。一旦把中间件压爆，可能整个交易系统。

持久化与恢复

防止系统崩溃导致的数据丢失。

事务消息

在需要保证数据一致性的场景中，合理使用事务消息。

消息确认与重试

合理设置消息确认机制和重试策略。如果本次无法处理，一定再抛回去。建议不要先确认再处理，万一确认后，本机又无法处理，消息就丢失了。

5. 支付系统应用案例

消息中间件在支付系统中核心有几个核心应用场景：

1. 流量的削峰填谷。尤其是支付交易。

2. 系统应用间的解耦。比如支付成功后，渠道网关发出支付成功消息，由支付引擎和账务分别监听。
3. 事务消息。
4. 离线数据同步。有些公司使用离线库来做，有些公司直接抛消息。比如实时库根据用户来分库分表，离线库要根据商户来聚合等。

基本上，消息中间件在支付系统中无所不在。

6. JAVA版的示例代码

先声明，以下代码纯演示，生产环境需要考虑更多因素。

比如支付场景下，接收请求后，先放到队列，然后使用单独的消费线程去消费。以下是简单示例。

1. 添加依赖

首先，你需要在项目的 pom.xml 文件中添加 RocketMQ 的依赖：

```xml
<dependencies>
  <dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-client</artifactId>
    <version>5.1.4</version>
  </dependency>
</dependencies>
```

确保使用最新版本的依赖。

2. 创建生产者 (Producer)

创建一个生产者以发送消息到 RocketMQ 服务器：

```java
public class PayServiceImpl implements PayService {
    @Autowired
    private MQProducer mqProducer;

    public PayOrder pay(PayRequest request) {
        PayOrder payOrder = buildPayOrder(request);
        // 前置处理，比如校验、保存DB等
        ... ...

        // 发送到队列
        Message msg = buildMessage(payOrder);
        mqProducer.send(msg);

        // 前置处理
        ... ...

        return payOrder;
    }

    public boolean processPay(PayOrder payOrder) {
        // 外发处理
        ... ...
    }
}
```

3. 创建消费者 (Consumer)

创建一个消费者来接收从 RocketMQ 发送的消息：

```java
public class PayConsumerServiceImpl implements PayConsumerService {
    @Autowired
    private MQConsumer mqConsumer;
    @Autowired
    private PayService payService;

    @Postconstruct
    public void init() {
        mqConsumer.registerMessageListener((MessageListenerConcurrently) (msgs, context) -> {
            payService.processPay(buildPayOrder(msg);
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        });
    }
}
```

再次声明，上述代码纯演示，在生产环境中，需要考虑更复杂的场景和错误处理机制。

7. 结束语

通过使用分布式消息中间件进行并发流量控制，支付系统可以更有效地应对高并发场景。能很好地提高系统整体的稳定性和可靠性，毕竟瞬间的大流量被缓冲到了消息中间件里。

但有些场景无法使用消息中间件，比如要求整个集群低到1TPS，又或者对接了外部上百个渠道，每个渠道要求不一样，有些要求最高20TPS，有些最高100TPS，使用消息中间件不好实现，就需要前面文章介绍的手撸一个漏桶或令牌桶。

下一篇聊聊阿里开源的流量控制与熔断利器：Sentinel。

这是《图解支付系统设计与实现》专栏系列文章中的第（18）篇。

深耕境内/跨境支付架构设计十余年，欢迎关注并星标公众号“隐墨星辰”，和我一起深入解码支付系统的方方面面。

专栏系列文章PDF合集不定时更新，欢迎关注我的公众号“隐墨星辰”，留言“PDF”获取。

隐墨星辰 公众号

10年顶尖境内/跨境支付公司架构经验

著有《图解支付系统设计与实现》

和我一起解码支付系统方方面面

有个支付系统设计与实现讨论群，添加个人微信（yinmon_sc）备注666进入。

隐墨星辰 个人微信

10年顶尖境内/跨境支付公司架构经验

著有《图解支付系统设计与实现》

备注666进支付讨论群