rocketMQ的个人理解和总结

rocketMQ是一款分布式，队列模型的消息中间件，MQ的主要特点为解耦、异步、削峰，具有高性能、高可靠、高实时、分布式特点，用于减少数据库压力的业务场景。

**rocketmq的组成：**

producer：消息生产者，将消息发送到mq.

producer group, 多个发送同一类消息的生产者，

consumer:消息消费者，消费MQ上的消息。

consumer group 消费者组，消费同一类消息的多个consumer。

topic 是一种消息的逻辑分类，对消息进行分类，

message 是消息的载体，一个message必须指定topic。

tag:标签是对topic进一步细化。相同业务中通过引入标签来标记不同用途的消息。

broker：rocketmq的主要角色，就是mq,接受来自生产者的消息，存储，以及为消费者拉去消息的请求做准备。

name server：为producer和consumer提供路由信息。

push consumer 应用向consumer注入一个listener接口，一旦收到消息，consumer立刻回调listener接口方法。push是客户端内部的回调机制。

pullconsumer consumer从服务端拉消息，然后处理。

多master多slave模式：异步复制。 主备有短暂延迟。

多master多slave模式：同步双写。主备都写成功了才返回成功。性能比异步复制略低。

**RecketMQ的原理：**

RocketMQ由NameServer注册中心集群、Producer生产者集群、Consumer消费者集群和若干Broker（RocketMQ进程）组成，它的架构原理是这样的：

Broker在启动的时候去向所有的NameServer注册，并保持长连接，每30s发送一次心跳

Producer在发送消息的时候从NameServer获取Broker服务器地址，根据负载均衡算法选择一台服务器来发送消息

Conusmer消费消息的时候同样从NameServer获取Broker地址，然后主动拉取消息来消费

**Broker、NameServer、Producer和Comsumer之间的关系：**

从 Broker 开始，Broker Master1 和 Broker Slave1 是主从结构，它们之间会进行数据同步，即 Date Sync。同时每个 Broker 与NameServer 集群中的所有节点建立长连接，定时注册 Topic 信息到所有 NameServer 中。  
 Producer 与 NameServer 集群中的其中一个节点（随机选择）建立长连接，定期从 NameServer 获取 Topic 路由信息，并向提供 Topic 服务的 Broker Master 建立长连接，且定时向 Broker 发送心跳。Producer 只能将消息发送到 Broker master，但是 Consumer 则不一样，它同时和提供 Topic 服务的 Master 和 Slave建立长连接，既可以从 Broker Master 订阅消息，也可以从 Broker Slave 订阅消息。

**RocketMQ的事务设计：**

应用模块遇到要发送事务消息的场景时，先发送prepare消息给MQ。  
prepare消息发送成功后，应用模块执行数据库事务（本地事务）。  
根据数据库事务执行的结果，再返回Commit或Rollback给MQ。  
如果是Commit，MQ把消息下发给Consumer端，如果是Rollback，直接删掉prepare消息。  
第3步的执行结果如果没响应，或是超时的，启动定时任务回查事务状态（最多重试15次，超过了 默认丢弃此消息），处理结果同第4步。  
MQ消费的成功机制由MQ自己保证。

**Rocketmq中生产者和消费者的负载均衡。**

生产者以轮询的方式向所有写队列发送消息，

在一个group中的消费者，可以以负载均衡的方式接收消息。

## 默认是使用**AllocateMessageQueueAveragely** 平均分配，将消息平均分配给群组内的每个消费者。

## **AllocateMessageQueueAveragelyByCircle 环形分配，**

在代码中加**consumer.setAllocateMessageQueueStrategy(new AllocateMessageQueueAveragelyByCircle())**

环形分配是指所有消息以此分配给每一个消费者。

## **AllocateMessageQueueConsistentHash 一致性哈希**

**在代码中加 consumer.setAllocateMessageQueueStrategy(new AllocateMessageQueueConsistentHash())**

这种算法依靠一致性哈希算法，看当前消费者可以落到哪个虚拟节点，该虚拟节点对应哪个队列。

**Rocketmq与springboot的整合**

**导入依赖：**

<dependency>

<groupId>org.apache.rocketmq</groupId>

<artifactId>rocketmq-spring-boot-starter</artifactId>

<version>${rocketmq-spring-boot-starter-version}</version>

</dependency>

配置：

spring.rocketmq.nameServer=120.56.195.135:9876;120.56.195.135:9877

spring.rocketmq.producer.group=lin-my-group

**生产者发送消息：**

public class Producer {

@Autowired

private RocketMQTemplate rocketMQTemplate;

public void test1(){

String msg="this is a rocketmq";

rocketMQTemplate.convertAndSend("springboot-mq",msg);

}

}

**消费者消费消息：**

@RocketMQMessageListener(topic = "springboot-mq",consumerGroup = "springboot-mq-consumer-1")

public class Consumer implements RocketMQListener<String>{

@Override

public void onMessage(String message) {

log.info("Receive message："+message);

}

}