数据库表设计

- tb_user：用户表
- tb_user_info：用户详情表
- tb_shop：商户信息表
- tb_shop_type：商户类型表
- tb_blog：用户日记表
- tb_comment：用户评论表
- tb_follow：用户关注表
- tb_voucher：优惠券表
- tb_seckill_voucher ： 秒杀优惠券表
- tb_voucher_order：优惠券的订单表

基于Session或Redis（优化）实现短信验证登录

利用session存储验证码，并返回sessionid给前端。再通过sessionid取回对应的session并从中读取出对应的验证码并校验。并利用拦截器进行登录验证。多台tomcat不共享session存储空间，**用Redis来解决session内存不共享的问题。用手机号作为key，验证码作为value实现验证码登录注册功能。利用token作为key，用户信息作为value实现登录验证校验登录状态**

利用redis解决token状态刷新问题

设置一个单独的拦截器，每次进行登录状态校验时刷新token的有效期。

使用Redis查询商铺缓存

提交商铺id，从Redis中根据id查询商铺缓存，判断缓存是否命中，若命中，则直接返回商铺信息；若未命中，则根据id查询数据库，再将查询到的数据写入Redis并返回。对于店铺的详细数据，变化比较大，店家会随时修改店铺的相关信息，所以对于这类变动比较频繁的数据，直接存入redis中并设置合适的有效期。

使用旁路缓存模式进行缓存的更新（读写）策略，**写：先修改数据库，然后删除缓存。读：先读缓存，若未命中，再读数据库。**

缓存穿透问题解决思路：当缓存未命中且数据库中没有相应数据时，将空值存入redis中并设置过期时间

缓存击穿问题：也叫热点Key问题，就是一个被高并发访问并且缓存重建业务较复杂的key突然失效了，无数的请求访问会在瞬间给数据库带来巨大的冲击

解决办法：

- 逻辑上添加过期时间，不主动设置过期时间。（推荐）
  根据id查询redis，判断缓存是否过期，如果过期，则获取互斥锁去数据库查询，若获取成功，则开启独立线程去数据库查询数据并写入redis并设置过期时间，若获取锁失败，则返回redis中的过期数据。数据在redis中过期后并不会被删除。
- 互斥锁解决缓存击穿问题（**不推荐**）：进行查询后如果缓存没有查询到数据，则进行互斥锁的获取，获取互斥锁后，判断是否获得了锁，如果没有获得，则休眠并稍后进行尝试，直到获得锁或redis中出现数据为止，才能进行查询。获取到锁的线程，对数据库进行查询，后将数据写入redis中，然后释放锁，返回数据。

缓存雪崩：同一时间段内大量的缓存key同时失效，导致大量的请求到达数据库，带来巨大压力。常见解决方案：

- 给不同的key的TTL添加随机值
- 利用reids集群增加服务的可用性

- 给缓存业务添加降级限流策略，
- 给业务添加多级缓存

使用Redis实现全局唯一ID

使用Redis的自增策略生成全局性唯一ID。根据当天的时间戳利用redis的increment生成计数并与以秒为单位的时间戳拼接生成订单号。

SETNX分布式锁

setnx指令只能设置key不存在的值，值不存在则设置成功，返回1；值存在则设置失败，返回0。使用lua脚本保证代码执行的原子性

超卖问题和一人一单问题

将耗时比较短的逻辑判断放入redis中，比如库存是否足够，比如是否一人一单，，然后在后台开一个线程，慢慢执行下单的操作

当用户下单之后，判断库存是否充足只需要到redis中去根据key找对应的value是否大于0即可，如果不充足，则直接结束，如果充足，继续在redis中判断用户是否可以下单，如果set集合中没有这条数据，说明他可以下单，如果set集合中没有这条记录，则将userId和优惠卷存入到redis中，并且返回0，整个过程需要保证是原子性的，我们可以使用lua来操作

当以上判断逻辑走完之后，我们可以判断当前redis中返回的结果是否是0 ，如果是0，则表示可以下单，则将之前说的信息存入到到queue中去，然后返回，然后再来个线程异步的下单，前端可以通过返回的订单id来判断是否下单成功。

Redission的分布式限流

RRateLimiter采用令牌桶和固定时间端口

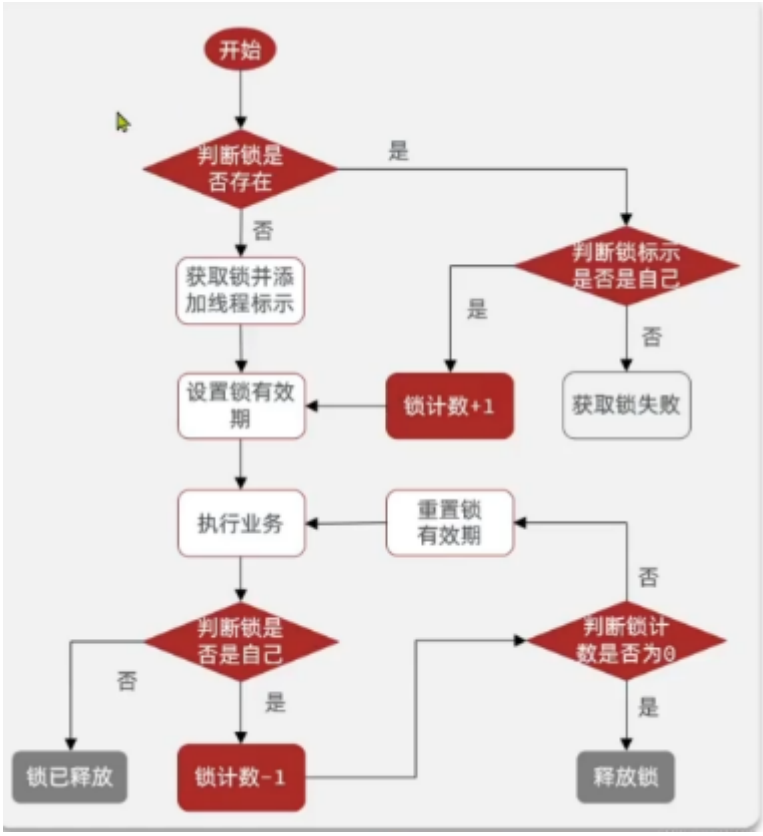
Redission原理

trylock方法介绍：

采用发布订阅机制，当获取锁的尝试失败时，会在指定时间内等待，等待期间接收到了释放锁的通知则再次尝试获取锁。

- trylock()：使用默认的超时时间和等待机制。具体的超时时间由Redission配置文件或者自定义配置决定
- trylock(long time,TimeUnit unit)：在指定时间内（等待time）尝试获取锁，如果成功则返回true，如果失败则等待后重试；指定时间内未能获取锁，则返回false。该参数未设置则尝试失败后直接返回。
- tryLock(long waitTime,long leaseTime,TimeUnit unit)：指定等待时间为waitTime，leaseTime为超时释放时间，即超过leasetime后还未释放锁则自动释放，默认超时时间为30s，同时使用超时续约机制，每隔一段时间进行自动续约。

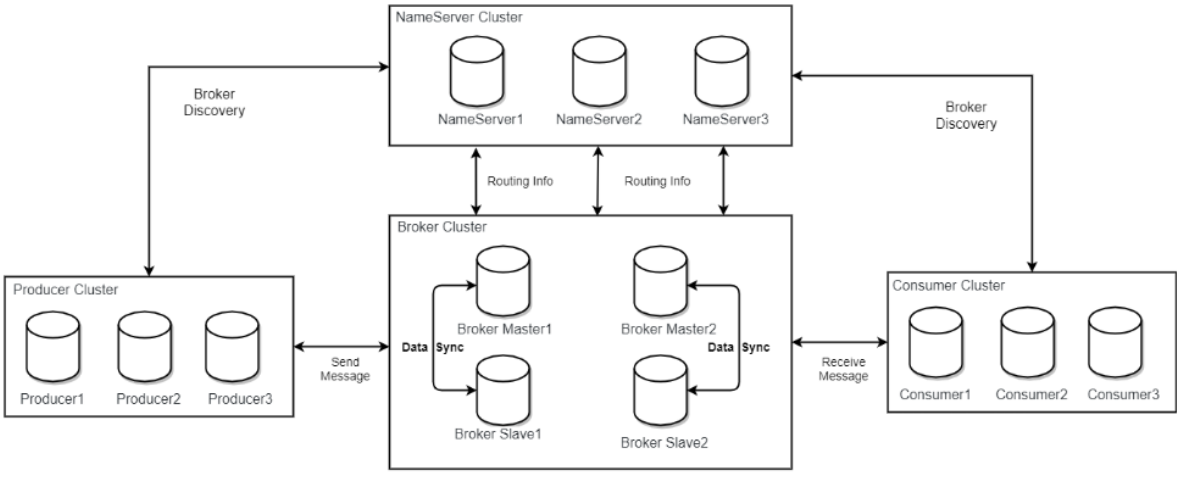
可重入锁原理：



Redission内部释放锁并不是直接执行del命令删除锁，而是将锁以hash数据结构的形式存储在redis中，每次获取锁则将value+1，释放锁，则将value-1，只有value值归零时才释放锁

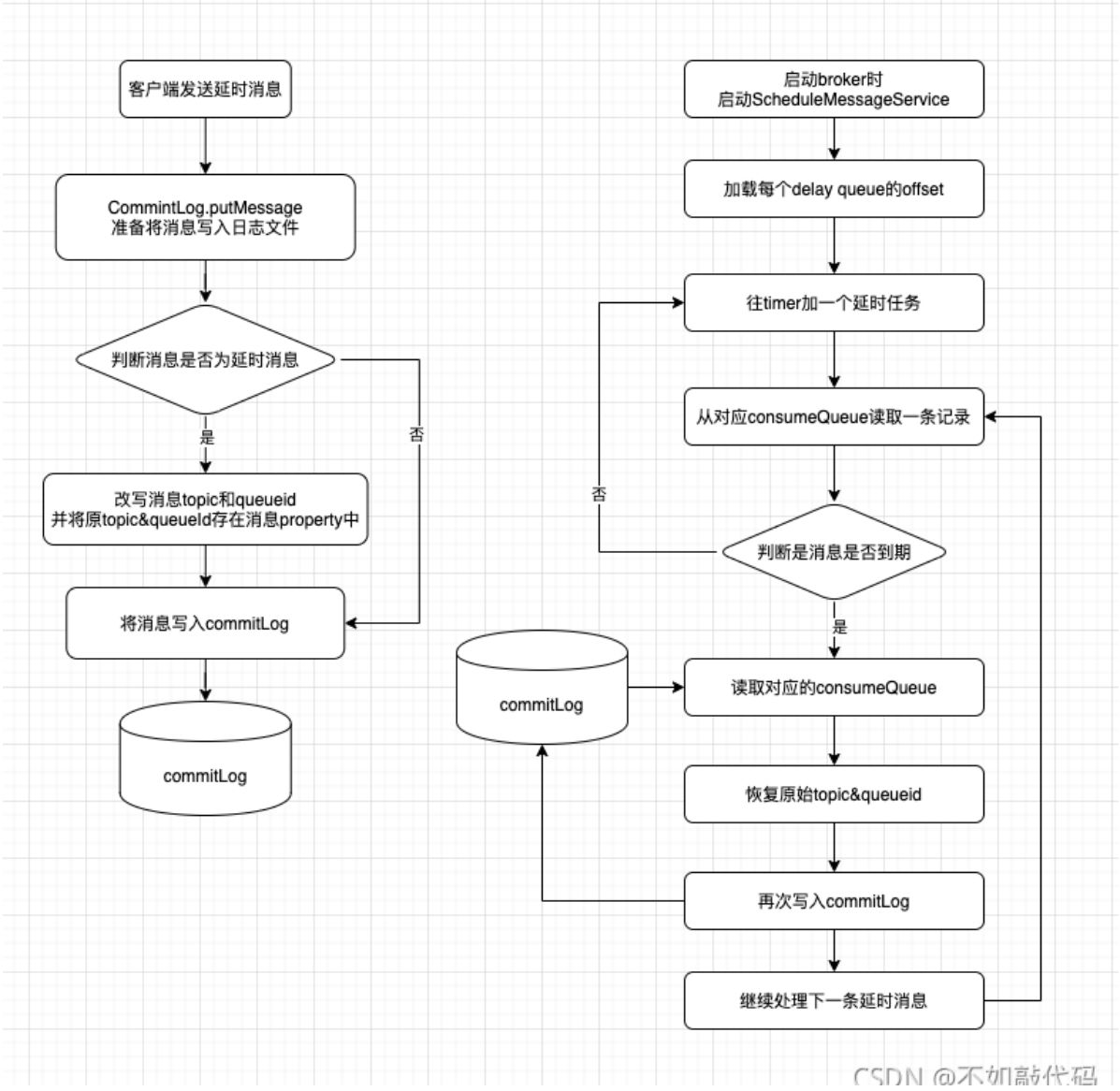
RocketMQ

整体架构：



- nameServer提供注册中心的服务，负责broker的管理，以及topic路由信息的管理。
- brokerServer则主要负责消息的存储、投递和查询及高可用
- Producer连接nameServer获取到broker的信息后，发送信息到对应的broker
- Consumer同样连接nameServer，查询topic路由信息，然后连接broker消费信息。

rocketmq的所有消息都存储在commitlog中，然后ConsumerQueue作为逻辑消费队列，维护一个topic消息的索引，记录topic内消息在commitlog中的一些信息。其中ConsumerQueue的存储单元为8字节的offset+4字节的size+8字节的tags hashcode，对于延时消息，最后8字节则用于存储消息计划投递时间。



rocketmq在最终将message存入commitlog中时，会先判断是否是延时消息，如果是延时消息则替换topic为SCHEDULE_TOPIC_XXXX，并将原topic存入message.properties中，之后根据指定延时level存入指定的queue。延时发送的代码，启动1s后，对已创建的delayQueue启动一个投递延时消息的任务，然后根据offet批量拉取对应的消息，判断是否到达投递时间，未到达则使用timer延时对应时长后启动下一次投递任务；到达投递时间后则回复原始的topic和queueid并调用writeMessageStore.putMessage(msgInner)将消息再次投递到commitlog中，然后投递下一个消息。commitLog是文件，消息存储在文件当中。

延时队列实现订单支付或取消的逻辑

使用一个rocketmq实现订单的创建工作并存入redis中，同时使用另一个延时rocketmq实现订单到期取消的监视功能。订单创建完成后若用户确认订单则将状态改为已确认，若取消则将状态改为关闭，延时队列到期后若该订单仍然处于创建中，则将订单改为已关闭。

Set实现点赞功能

不使用Mysql而是使用Redis的Set统计日记的点赞数据

SortedSet实现点赞排行榜

点赞时的时间戳作为score实现点赞排序功能。

好友关注和取关

在好友关注表内记录关注数据，利用Set实现共同关注的数据筛选功能。

评论的结构设计

将评论分为根评论和子评论，根评论即为直接对日记的评论，子评论即为对评论的回复。

名	类型	长度	小数点	不是 null	键	注释
videoCommentId	bigint	20	0	☑	🔑1	评论id
content	varchar	1000	0	☑		内容
createTime	datetime	0	0	☐		评论时间
isDelete	bit	1	0	☑		是否删除
userId	int	11	0	☑		所属用户id
videoId	bigint	20	0	☑		所属视频id
commentLikeCount	int	11	0	☐		点赞次数
rootCommentId	bigint	20	0	☐		顶级评论id
toCommentId	bigint	20	0	☐		回复目标评论id
updateTime	datetime	0	0	☐		修改时间

给表添加一个rootCommentId字段，如为null，则该评论为顶级评论，否则为顶级评论的id。
给表添加一个toCommentId字段，如为null则为顶级评论，否则为目标评论的id。

Feed流实现方案

拉模式

也叫做读扩散。当张三李四王五发了信息之后，会保存在自己的信箱，赵六读取信息时，会读取自己的收件箱，此时系统会从他关注的人群中，把关注人的信息全部进行拉取，然后进行排序。
优点：节约空间，只有在读取信息时才进行消息的接收
缺点：比较延迟，对服务器压力大

拉模式

也叫做写扩散。没有邮箱，张三写了一个内容后，会直接发送到他的粉丝收件箱中，不再需要临时拉取
优点：时效快，不用临时拉取
缺点：内存压力大，信息会复制很多份

推拉结合模式

读写混合。普通人写入到他的粉丝中，大V留在自己的邮箱中，然后再写一份只给他的活跃粉丝

推送实现

大V有自己的发件箱，每个人有自己的收件箱。都是用ZSet实现，存储blogid和时间戳。普通人发blog后会推送给自己的所有粉丝，大V发blog后只推荐给自己的活跃粉丝，然后保存一份在自己的发件箱中。

用户签到和连续签到统计

使用bitmap实现用户签到和签到统计功能