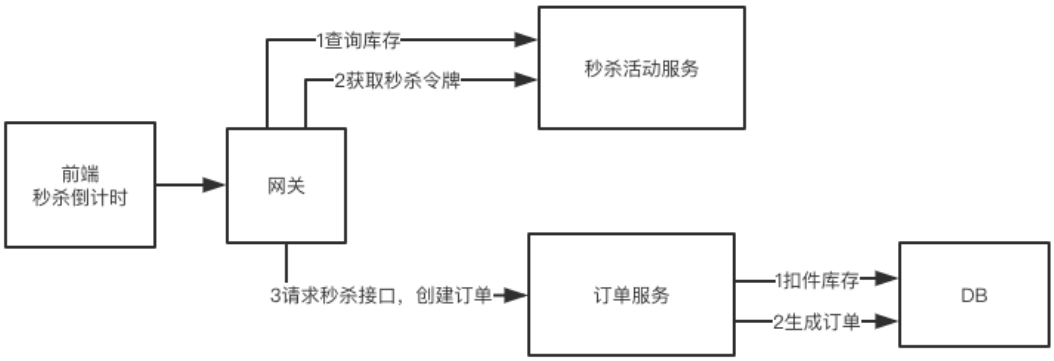
影院项目（Redis、RocketMQ、Nginx 等）资料问题整理

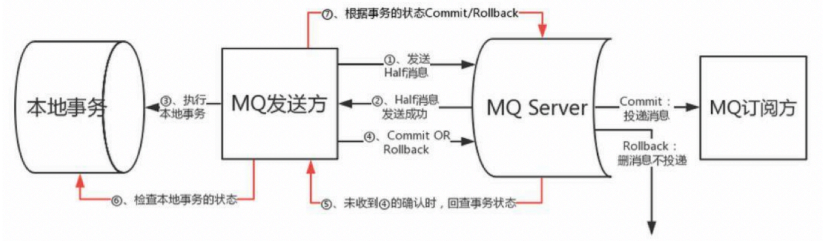
1.1 说一说秒杀过程?



扣减库存的是时候，由于mysql性能限制，所以把库存存入redis中，此时会产生一个redis中库存和数据库中库存一致性的问题，我们这里采用rocketMQ事务型消息来保证数据一致性

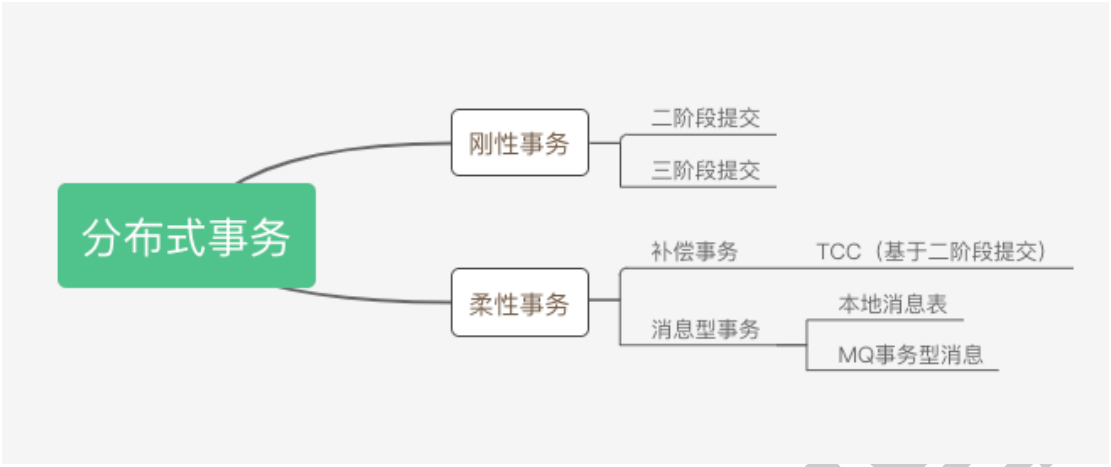
1.2 说一说 RocketMQ 如何实现分布式事务?

把下面这个图的流程讲解清楚

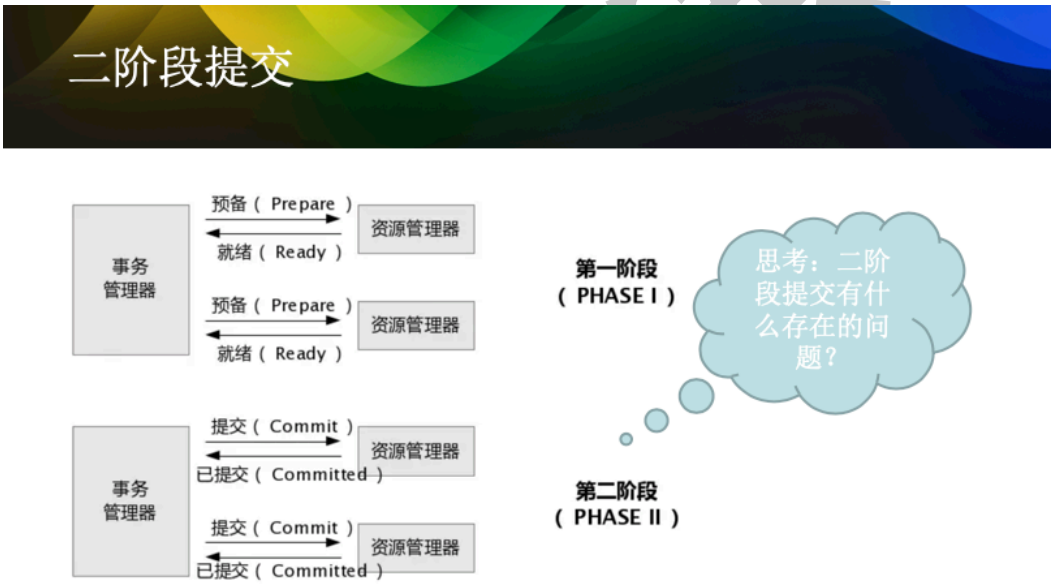


1，事务发起方发送prepare消息到MQ
2，消息发送成功后执行本地事务
3，根据本地事务执行结果返回commit或者是rollback
4，如果消息时rollback，MQ将删除该prepare消息不进行下发，如果是commit消息，mq将会把这个消息发送到consumer端
5，执行本地事务的过程中，执行端挂掉，或者超时，MQ将会不停的询问其同组的其他producer来获取状态
6，Consumer端的消费成功机制有MQ保证

1.3 分布式事务有哪几种？简单说一下二阶段提交



二阶段提交：



二阶段提交把事务分为两个阶段，并引入了事务管理器

阶段一：

1. 提交事务请求：协调者向所有的参与者发送事务内容，询问是否可以执行事务提交操作，并开始等待各参与者的响应。
2. 执行事务：各个参与者节点执行事务操作。并将 Undo 和 Redo 信息记入事务日志中。
3. 各参与者向协调者反馈事务询问的响应：如果参与者成功执行了事务操作，那么就反馈给协调者 yes 响应，表示事务可以执行，如果参与者没有成功执行事务，那么

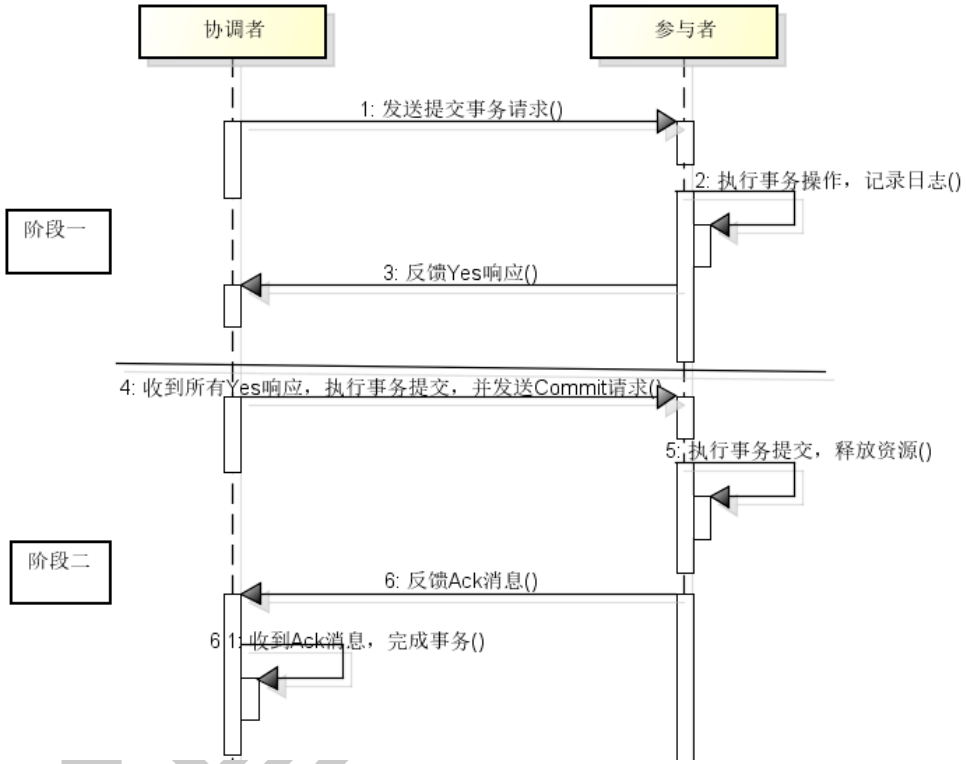
就反馈给协调者 no 响应，表示事务不可以执行。

阶段二：

1，执行事务提交：如果事务管理器从所有事务参与者处获得的响应都为 Yes，那么就会执行事务提交，协调者向所有参与者节点发出 commit 请求。

2，参与者提交事务：参与者接收到 commit 请求后，会正式执行事务提交的操作，并在完成提交之后释放整个事务执行期间占有的事务资源，并向协调者发送 ack 消息

3，完成事务：协调者接收到所有参与者反馈的 Ack 消息后，完成事务。



如上图，如果第四步收到 No 响应，或者是等待操作，没有收到所有响应，进入事务中断，然后事务协调者发送 Rollback 请求，请求资源回滚，释放资源

1.4 说一说 Redis 的持久化策略，你们项目用的是哪一种?

Redis 的持久化方式有俩种，持久化策略有 4 种：

RDB（数据快照模式），定期存储，保存的是数据本身，存储文件是紧凑的
AOF（追加模式），每次修改数据时，同步到硬盘(写操作日志)，保存的是数据的变更记录
如果只希望数据保存在内存中的话，俩种策略都可以关闭
也可以同时开启俩种策略，当 Redis 重启时，AOF 文件会用于重建原始数据

1.5 Redis 的数据类型有哪几种？知道 setnx 和 setex 是干嘛的吗？

Redis 支持五种数据类型 :string (字符串)，hash (哈希)，list (列表)，set (集合) 及 zset(sorted set：有序集合)。

Setnx: 设置 key 对应的值为 string 类型的 value。 如果 key 已经存在，返回 0，nx 是 not exist 的意思，可用于分布式锁。例如：setnx key value

Setex: 设置 key 对应的值为 string 类型的 value，并指定此键值对应的有效期。例如：setex key seconds value

1.6 消息队列都有哪些？你们项目用的 rocketMQ 对比其他消息队列有什么优势？

特性	ActiveMQ	RabbitMQ	Rocket MQ	kafka
开发语言	java	erlang	java	scala
单机吞吐量	万级	万级	10万级	10万级
时效性	ms级	us级	ms级	ms级以内
可用性	高(主从架构)	高(主从架构)	非常高(分布式架构)	非常高(分布式架构)
功能特性	成熟的产品，在很多公司得到应用；有较多的文档；各种协议支持较好	基于erlang开发，所以并发能力很强，性能极其好，延时很低;管理界面较丰富	MQ功能比较完备，扩展性佳	只支持主要的MQ功能，像一些消息查询，消息回溯等功能没有提供，毕竟是为大数据准备的，在大数据领域应用广。

1.7 说一下分布式锁？

单机环境下，我们用 Synchronized 关键字来修饰方法或者是修饰代码块，以此来保证方法

内或者是代码块内的代码在同一时间只能被一个线程执行。但是如果到了分布式环境中，Synchronized 就不好使了，因为他是 JVM 提供的关键字，只能在一台 JVM 中保证同一时间只有一个线程执行，那么假如现在有多台 JVM 了（项目有多个节点了），那么就需要分布式锁来保证同一时间，在多个节点内，只有一个线程在执行该代码。

三种实现方式：

1 基于数据库：

数据库自带的通过唯一性索引或者主键索引，通过它的唯一性，当某个节点来使用某个方法时就在数据库中添加固定的值，这样别的节点来使用这个方法就会因为不能添加这个唯一字段而失败。这样就实现了锁。一个节点执行完毕后就删除这一行数据，别的节点就又能访问这个方法了，这就是释放锁。但是一方面，数据库本身性能就低，而且这种形式非常容易死锁，虽然可以各种优化，但还是不推荐使用。数据库的乐观锁也是一个道理，一般不用

2 Redis 分布式锁

SETNX 关键字，设置参数如果存在返回 0，如果不存在返回 value 和 1
expire 关键字，为 key 设置过期时间，解决死锁。
delete 关键字，删除 key，释放锁
实现思想：
（1）获取锁的时候，使用 setnx 加锁，并使用 expire 命令为锁添加一个超时时间，超过该时间则自动释放锁，锁的 value 值为一个随机生成的 UUID，通过此在释放锁的时候进行判断。
（2）获取锁的时候还设置一个获取的超时时间，若超过这个时间则放弃获取锁。
（3）释放锁的时候，通过 UUID 判断是不是该锁，若是该锁，则执行 delete 进行锁释放。

zookeeper 实现分布式锁
ZooKeeper 是一个为分布式应用提供一致性服务的开源组件，它内部是一个分层的文件系统目录树结构，规定同一个目录下只能有一个唯一文件名。基于 ZooKeeper 实现分布式锁的步骤如下：
（1）创建一个目录 mylock；
（2）线程 A 想获取锁就在 mylock 目录下创建临时顺序节点；
（3）获取 mylock 目录下所有的子节点，然后获取比自己小的兄弟节点，如果不存在，则说明当前线程顺序号最小，获得锁；
（4）线程 B 获取所有节点，判断自己不是最小节点，设置监听比自己次小的节点；
（5）线程 A 处理完，删除自己的节点，线程 B 监听到变更事件，判断自己是不是最小的节点，如果是则获得锁。
速度没有 redis 快，但是能够解决死锁问题，可用性高于 redis

1.8 你们项目哪里用了线程池？说一下线程池，几种配置方式？缺点是什么？

例如：Executors.newCacheThreadPool()，是 JUC 包提供的线程池的创建方式，可以对照 Java 源码自己看一下

newCachedThreadPool：

底层：返回 ThreadPoolExecutor 实例，corePoolSize 为 0；maximumPoolSize 为 Integer.MAX_VALUE；keepAliveTime 为 60L；unit 为 TimeUnit.SECONDS；workQueue 为 SynchronousQueue(同步队列)

通俗：当有新任务到来，则插入到 SynchronousQueue 中，由于 SynchronousQueue 是同步队列，因此会在池中寻找可用线程来执行，若有可以线程则执行，若没有可用线程则创建一个线程来执行该任务；若池中线程空闲时间超过指定大小，则该线程会被销毁。

适用：执行很多短期异步的小程序或者负载较轻的服务器

newFixedThreadPool：

底层：返回 ThreadPoolExecutor 实例，接收参数为所设定线程数量 nThread，corePoolSize 为 nThread，maximumPoolSize 为 nThread ；keepAliveTime 为 0L(不限时) ；unit 为：TimeUnit.MILLISECONDS；WorkQueue 为：new LinkedBlockingQueue<Runnable>() 无解阻塞队列

通俗：创建可容纳固定数量线程的池子，每隔线程的存活时间是无限的，当池子满了就不在添加线程了；如果池中的所有线程均在繁忙状态，对于新任务会进入阻塞队列中(无界的阻塞队列)

适用：执行长期的任务，性能好很多

newSingleThreadExecutor:

底层：FinalizableDelegatedExecutorService 包装的 ThreadPoolExecutor 实例，corePoolSize 为 1；maximumPoolSize 为 1；keepAliveTime 为 0L；unit 为：TimeUnit.MILLISECONDS；workQueue 为：new LinkedBlockingQueue<Runnable>() 无解阻塞队列

通俗：创建只有一个线程的线程池，且线程的存活时间是无限的；当该线程正繁忙时，对于新任务会进入阻塞队列中(无界的阻塞队列)

适用：一个任务一个任务执行的场景

NewScheduledThreadPool:

底层：创建 ScheduledThreadPoolExecutor 实例，corePoolSize 为传递来的参数，maximumPoolSize 为 Integer.MAX_VALUE；keepAliveTime 为 0；unit 为：TimeUnit.NANOSECONDS；workQueue 为：new DelayedWorkQueue() 一个按超时时间升序排序的队列

通俗：创建一个固定大小的线程池，线程池内线程存活时间无限制，线程池可以支持定时及周期性任务执行，如果所有线程均处于繁忙状态，对于新任务会进入 DelayedWorkQueue 队列中，这是一种按照超时时间排序的队列结构

适用：周期性执行任务的场景

线程池任务执行流程：

1，当线程池小于 corePoolSize 时，新提交任务将创建一个新线程执行任务，即使此时线程池中存在空闲线程。

2，当线程池达到 corePoolSize 时，新提交任务将被放入 workQueue 中，等待线程池中任务调度执行

3，当 workQueue 已满，且 maximumPoolSize>corePoolSize 时，新提交任务会创建新

线程执行任务

4，当提交任务数超过 maximumPoolSize 时，新提交任务由 RejectedExecutionHandler 处理

5，当线程池中超过 corePoolSize 线程，空闲时间达到 keepAliveTime 时，关闭空闲线程

6，当设置 allowCoreThreadTimeOut(true)时，线程池中 corePoolSize 线程空闲时间达到 keepAliveTime 也将关闭

1.9 说一下 zookeeper 的作用？zookeeper 在某一个瞬间挂了 dubbo 服务还能调用吗？如何实现 zookeeper 的高可用？

作用：作为注册中心为分布式环境下各个微服务之间的调用提供协调工作
问题二：可以的，启动 dubbo 时，消费者会从 zk 拉取注册的生产者的地址接口等数据，缓存在本地。每次调用时，按照本地存储的地址进行调用
实现高可用：

搭建 zookeeper 集群实现高可用。

1.10 zookeeper 集群最少需要配置几台机器？为什么？

最少：3 台
Zookeeper 关于 leader 的选举机制，主要提供了三种方式：
LeaderElection
AuthFastLeaderElection
FastLeaderElection

默认的算法是 FastLeaderElection
在 zookeeper 的选举过程中，为了保证选举过程最后能选出 leader，就一定不能出现两台机器得票相同的僵局，所以一般的，要求 zk 集群的 server 数量一定要是奇数，也就是 2n+1 台，并且，如果集群出现问题，其中存活的机器必须大于 n+1 台，否则 leader 无法获得多数 server 的支持，系统就自动挂掉。所以一般是 3 个或者 3 个以上节点。

1.12 数据库中有多少条记录？考虑过分库分表吗？

根据阿里《Java 开发手册》提出的单表行数超过 500w 行或者是单表容量超过 2GB，推荐使用分库分表来看，mysql 单表性能瓶颈在 500w 左右（具体也与表结构设计和机器配置有关）

分库分表方案：

分库：使用 mycat 或者是 Sharing-JDBC 等中间件来作为数据库的代理

分表：单表业务量过大，可以采用按业务分表或者是按时间分表等手段来切分

分库分表，结合实际需求来操作，一般来说在一个项目的初期是不会考虑分库分表的，随着业务量和数据量的不断上升，才有一些性能上的要求，这个时候假如数据库有瓶颈了就可以考虑使用分库分表来提高系统的性能。

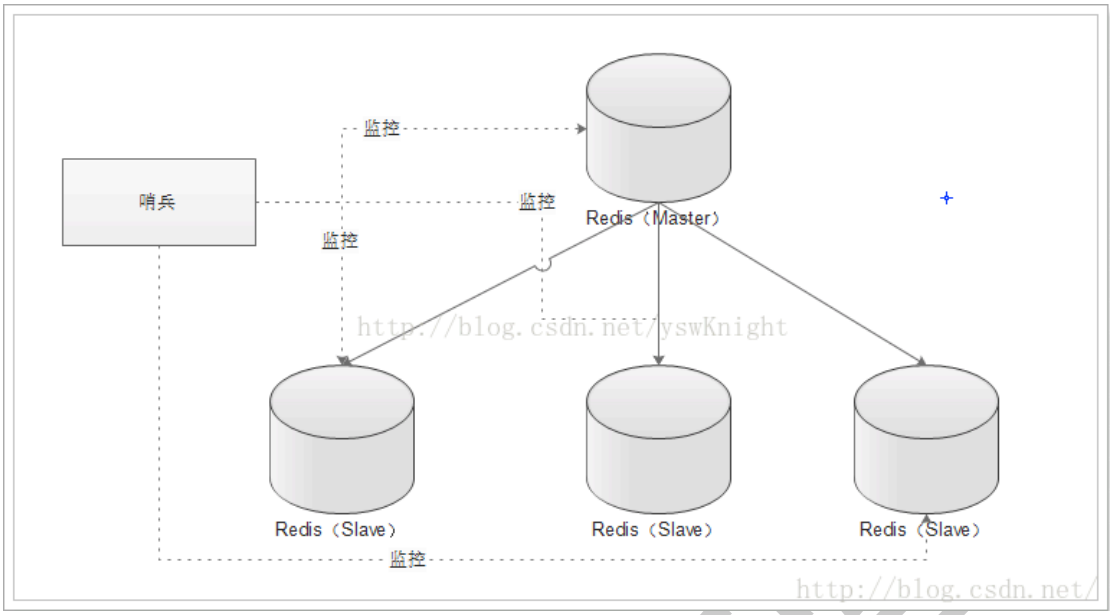
1.13 Redis 是如何配置的？说一下 redis 的集群模式、哨兵模式

问题 1:
主要说下 redis 的配置文件里面的配置,说最重要的即可。下面是几个比较重要的配置信息：

1，port:6379 指定访问 redis 服务的端口
2，bind：0.0.0.0 指定 redis 绑定的主机地址，配置为 0.0.0.0 表示为外网也可以访问
3，timeout：指定客户端连接 redis 服务器时，当闲置的时间为多少时关闭连接
4，loglevel：指定 redis 数据库的日志级别，常用的日志级别有 debug、verbose、notice、warning，不进行修改的情况下默认的是 notice
5，save <s> <c>：指定 redis 数据库多少时间内（s）就有多少次<c>更新操作（insert、update 和 delete）时就把缓存中的数据同步到本地库，比如 save 600 2，指的是 10 分钟内有 2 次更新操作，就同步到本地库，也就是持久化到本地磁盘。
6，dir 指本地数据文件存放的目录
7，requirepass：指定 redis 的访问密码
8，maxmemory：指定 redis 的最大内存。Redis 在启动时会把数据加载到内存中，当到达最大内存时，redis 会自动清除已经到期和即将到期的 key 值
9，appendonly：指定 redis 是否开启日志记录功能。由于 redis 时利用 save 命令异步的方式更新数据到本地库，所以不开启日志记录功能，可能会导致在出现生产事故时，导致部分数据未更新到本地库。
10，vm-enabled: 指定 redis 是否启用虚拟内存机制

redis 的集群模式：

1，主从复制模式：
Slave 从节点服务启动并连接到 Master 之后，它将主动发送一个 SYNC 命令。Master 服务主节点收到同步命令后将启动后台存盘进程，同时收集所有接收到的用于修改数据集的命令，在后台进程执行完毕后，Master 将传送整个数据库文件到 Slave，以完成一次完全同步。而 Slave 从节点服务在接收到数据库文件数据之后将其存盘并加载到内存中。此后，Master 主节点继续将所有已经收集到的修改命令，和新的修改命令依次传送给 Slaves，Slave 将在本次执行这些数据修改命令，从而达到最终的数据同步。
2，哨兵模式：



哨兵(sentinel) 是一个分布式系统,你可以在一个架构中运行多个哨兵(sentinel) 进程,这些进程使用流言协议(gossipprotocols)来接收关于 Master 是否下线的信息,并使用投票协议(agreement protocols)来决定是否执行自动故障迁移,以及选择哪个 Slave 作为新的 Master。

每个哨兵(sentinel) 会向其它哨兵(sentinel)、master、slave 定时发送消息,以确认对方是否" 活" 着,如果发现对方在指定时间(可配置)内未回应,则暂时认为对方已挂(所谓的" 主观认为宕机" Subjective Down,简称 sdown).

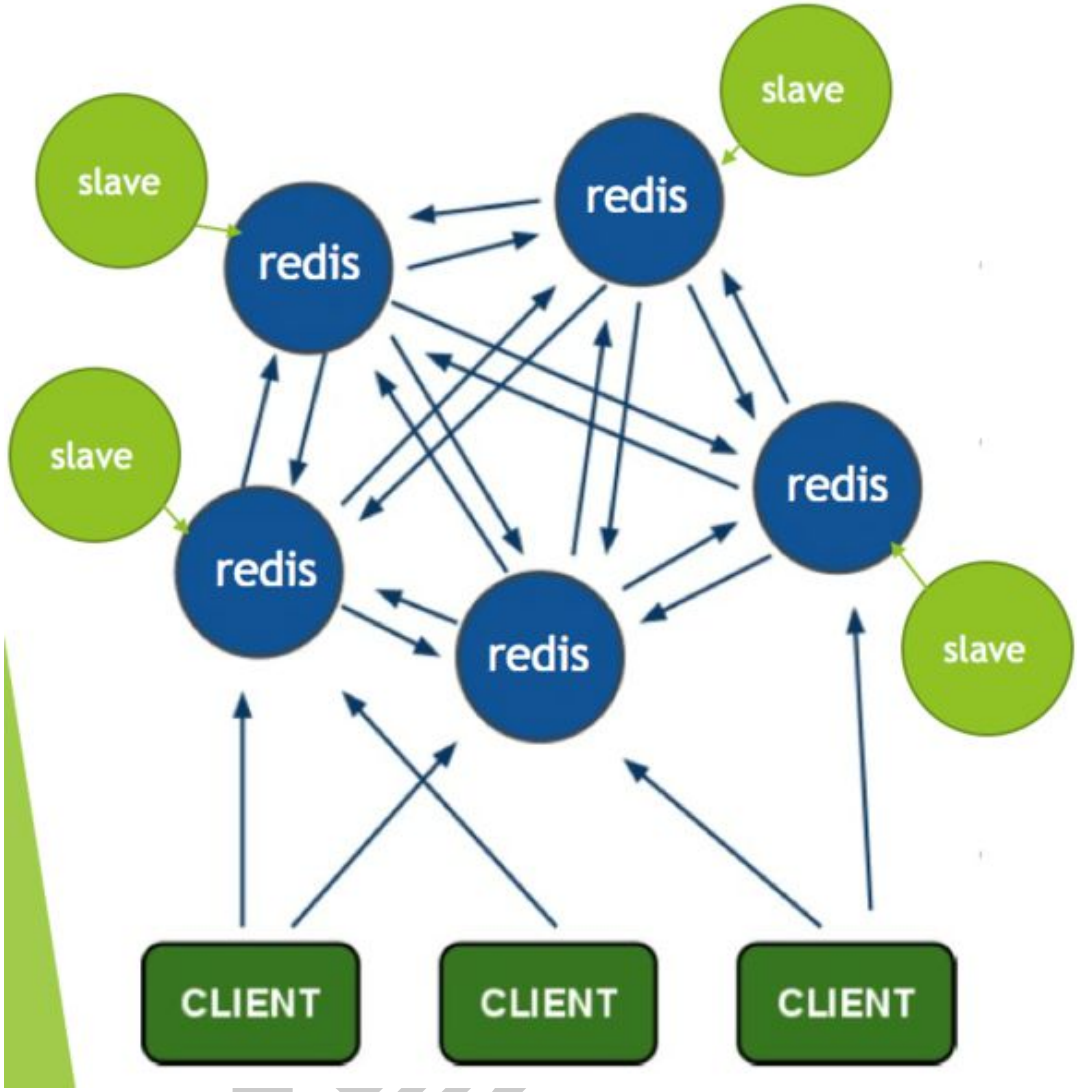
若 "哨兵群" 中的多数 sentinel,都报告某一 master 没响应,系统才认为该 master"彻底死亡"(即:客观上的真正 down 机,Objective Down,简称 odown),通过一定的 vote 算法,从剩下的 slave 节点中,选一台提升为 master,然后自动修改相关配置。

虽然哨兵(sentinel) 释出为一个单独的可执行文件 redis-sentinel ,但实际上它只是一个运行在特殊模式下的 Redis 服务器，你可以在启动一个普通 Redis 服务器时通过给定 --sentinel 选项来启动哨兵(sentinel)。

3， Cluster 集群模式：

自动将数据进行分片，每个节点上放一部分数据，如何放的呢？

Redis-Cluster 采用无中心结构，每个节点保存数据和整个集群状态，每个节点都和其他所有节点连接，其结构图如如下：



结构特点：

1，所有的 redis 节点彼此互联（PING-PONG 机制），内部使用二进制协议优化传输速度和带宽。
2，节点的 fail 是通过集群中超过半数的节点检测失效时才失效。
3，客户端与 redis 节点直连，中间不需要 proxy 层，客户端不需要连接所有集群的所有节点，连接集群中任意一个可用节点即可。
4，Redis-cluster 把所有的物理节点映射到 16384 个哈希槽（hash slot）上，不一定是平均分配，cluster 负责维护 node<->slot<->value。
5，当有数据需要存储的时候，redis 通过 CRC16 算法计算出 key 的值，然后把这个值对 16384 取余，得到的结果在哪个节点负责的范围内就会把值存入哪个节点。
6，如上图，每个 master 都是一个节点，而 slave 是该 master 的备份，当 redis 的一个 master 节点挂了，slave 会顶替掉 master，如何一个节点的 master 和 slave 都挂了，那么则认为该 redis 集群挂了，因为有一部分数据丢失了，也有一部分数据永远也存取不了了。

1.14 如何限流?

1，应用级别

A，限制瞬时并发数 ，如 nginx 的 limit_conn 模块，用来限制瞬时并发连接数、nginx 的 limit_req 模块，限制每秒的平均速率。

B，限流总并发/连接/请求数，如配置 tomcat 的线程池的参数：

acceptCount:如果 Tomcat 的线程都忙于响应，新来的连接会进入队列排队，如果超出排队大小，则拒绝连接

maxConnections:瞬时最大连接数，超出的会排队等待

maxThreads:Tomcat 能启动用来处理请求的最大线程数，如果请求处理量一直远远大于最大线程数则可能会僵死

2，服务级别

A，限制总资源数

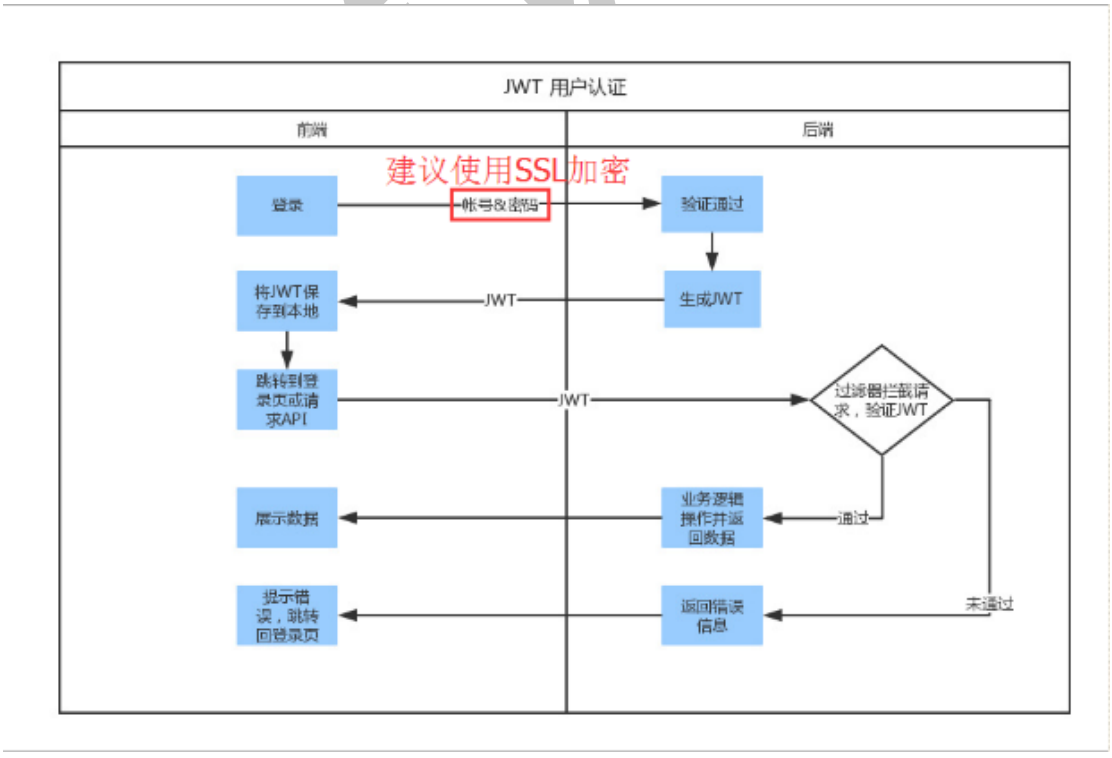
使用线程池，连接池等等，比如分配给每个应用的数据库连接是 100 个，那么该应用在同时最多能使用 100 个资源，超出了可以等待或者抛出异常

B，限流某个接口

a) 限制某个接口的总并发量：秒杀大闸，信号量，AtomicLong 等等

b) 平滑限流某个接口的请求数：令牌桶，RateLimiter

1.15 JWT 的原理?



1）前端通过表单将用户名和密码发送到后端接口。

2）后端核对用户名和密码后，将用户的 id 及其他非敏感信息作为 JWT Payload，将其与头部分别进行 base64 编码后签名，生成 JWT

3)后端将 JWT 字符串作为登录成功的结果返回给前端，前端可以将 JWT 存到 localStorage 或者 sessionStorage 中，退出登录时，前端删除保存的 JWT 信息即可。

4）前端每次在请求时，将 JWT 放到 header 中的 Authorization

5）后端验证 JWT 的有效性

6）验证通过后，进行其他逻辑操作。

1.16 Nginx 了解吗？会部署吗？如何做负载均衡？

1、Nginx 的负载分发策略

Nginx 的 upstream 目前支持的分配算法：

1)、轮询 ——1：1 轮流处理请求（默认）

每个请求按时间顺序逐一分配到不同的应用服务器，如果应用服务器 down 掉，自动剔除，剩下的继续轮询。

2)、权重

通过配置权重，指定轮询几率，权重和访问比率成正比，用于应用服务器性能不均的情况。

3)、ip_哈希算法

每个请求按访问 ip 的 hash 结果分配，这样每个访客固定访问一个应用服务器，可以解决 session 共享的问题。

2、配置 Nginx 的负载均衡与分发策略

通过在 upstream 参数中添加的应用服务器 IP 后添加指定参数即可实现

1.17 说一说 dubbo 的异步调用？

用法：

1， 首先开启异步调用配置，假如是在 SSM 项目中，应该添加消费<dubbo:method>

通过 async=" true" 标识。假如是在 SpringBoot 项目中，应该添加注解 @Service(interfaceClass = UserApi.class,async = true)

2， 调用用户服务的接口的 UserAPI 接口的时候，会异步调用，我们直接调用并不会立马给返回结果，需要去主动获取，代码如下：

```
UserDO userDO = UserAPI.register(UserVO);
//此时 userDO 为空，因为是异步调用，不会立马去返回
Future<UserDO> future = RpcContext.getContext().getFuture();
UserDO userDO = future.get();
//这个时候才会真正的取到 userDO 对象
```

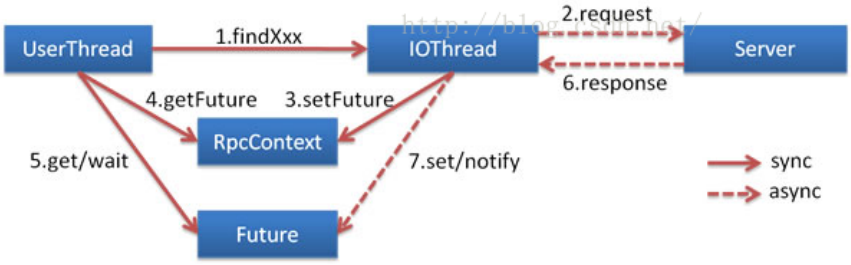
流程图：

异步调用

(+) (#)

基于NIO的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小。

2.0.6及其以上版本支持



1.18 JWT 的实现原理？如何做到安全？

见 1.15

如何做到安全？思路：

1， 生产 JWT 的加密算法和密钥（假如有的话）不要泄漏

2， 使用 HTTPS 传输，防止传输数据被拦截

1.19 说下 redis 的缓存雪崩，缓存穿透？如何保证 redis 缓存里面的都是热点数据？

缓存穿透：是指查询一个数据库一定不存在的数据。正常的使用缓存流程大致是，数据查询

先进行缓存查询，如果 key 不存在或者 key 已经过期，再对数据库进行查询，并把查询到的对象，放进缓存。如果数据库查询对象为空，则不放进缓存。那么如果查询的这个数据一直在缓存中不存在，那么就会一直 查询数据库，如果这个查询量很大的话，就有可能击穿我们的数据库，这个就叫缓存穿透。如何避免？当查询数据为空的时候，也可以在缓存中打上一个标记，这样下次就会去查缓存并且得知该数据为空了。

缓存雪崩：是指在某一段时间，缓存集中失效。假如大量缓存在某一个时刻同时失效，这个时候就会导致大量的请求去请求数据库，可能会引起数据库瘫痪。举例：
马上就要到双十二零点，很快就会迎来一波抢购，这波商品时间比较集中的放入了缓存，假设缓存一个小时。那么到了凌晨一点钟的时候，这批商品的缓存就都过期了。而对这批商品的访问查询，都落到了数据库上，对于数据库而言，就会产生周期性的压力波峰。

如何避免？设置热点缓存的时候过期时间分散设置。

如何保证 redis 里面都是热点数据？
当 redis 使用的内存超过了设置的最大内存时，会触发 redis 的 key 淘汰机制，在 redis 3.0 中有 6 种淘汰策略：
1，noeviction: 不删除策略。当达到最大内存限制时，如果需要使用更多内存,则直接返回错误信息。（redis 默认淘汰策略）
2，allkeys-lru: 在所有 key 中优先删除最近最少使用(less recently used ,LRU) 的 key。
3，allkeys-random: 在所有 key 中随机删除一部分 key。
4，volatile-lru: 在设置了超时时间（expire ）的 key 中优先删除最近最少使用(less recently used ,LRU) 的 key。
4， volatile-random: 在设置了超时时间（expire）的 key 中随机删除一部分 key。
6，volatile-ttl: 在设置了超时时间（expire ）的 key 中优先删除剩余时间(time to live,TTL) 短的 key。

我们可以把 redis 的淘汰策略设置为 LRU 策略（或者是别的策略，视具体业务场景而定）

1.20 如何保证你选的座位只有你一个人买？说一下分布式锁的实现方式

如何保证座位只有一个人买？
假如现在两个线程同时进来下单，座位号都是同一个座位。这个时候他们先去验证这个座位是否被买走了，然后发现这两个座位都没有被买走，这个时候他们同时去下单，然后我们需要在下单的逻辑那里加上分布式锁，来保证在同一时刻，只有一个线程能够对同一个座位下单。

分布式锁的实现方式：
3， 基于数据库实现分布式锁
   a) 核心思想：在数据库中创建一个表，表中包含方法名等字段，并在方法名字段上创建唯一索引，想要执行某个方法，就使用这个方法名向表中插入数据，成功插入则获取锁，执行完成后删除对应的行数据释放锁。

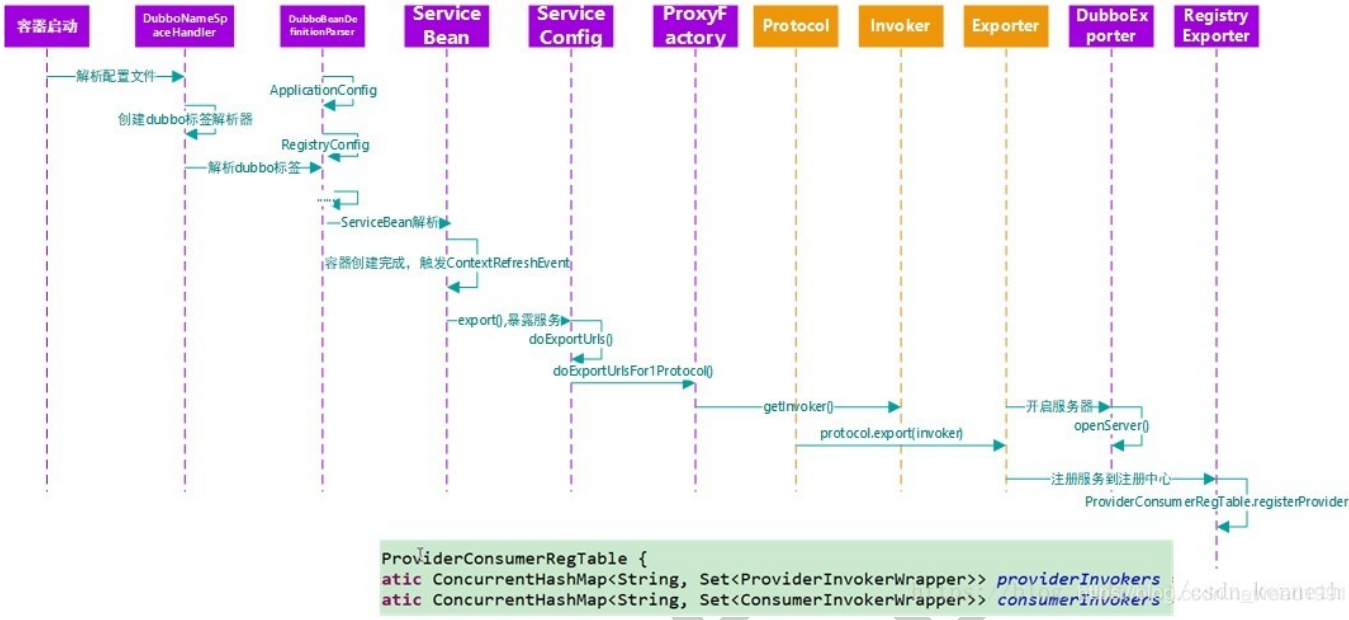
b) 优缺点：
   i. 基于数据库实现，性能差。
   ii. 不具备阻塞特性。
   iii. 不可重入

4，基于 redis(或者是 memcache)实现分布式锁（以 redis 为例，memcache 一样可以做）
a) 核心思想：获取锁的时候，使用 setnx 命令加锁，并为锁添加一个超时时间，超过该时间自动释放锁。
b) 优缺点：
   i. 性能好，实现方便
   ii. 容易死锁

5，基于 Zookeeper 实现分布式锁
a) 核心思想：根据有序节点+watch 实现，为每个线程生成一个有序的临时节点，为确保有序性，在排序一次全部节点，获取全部节点，每个线程判断自己是否最小，如果是的话，获得锁，执行操作，操作完删除自身节点。如果不是第一个的节点则监听它的前一个节点，当它的前一个节点被删除时，则它会获得锁，以此类推。
b) 优缺点：
   i. 不易死锁
   ii. 性能不如 redis
   iii. 可靠性好

1.21 说一下 dubbo 的注册（发现）流程？（涉及到 dubbo 的底层原理，建议有能力的同学可以去看下源码）

注册流程：



1，首先是容器启动的时候，会去解析配置文件，看是否需要解析配置 dubbo
2，创建 dubbo 标签解析器
3，解析 dubbo 标签并把配置的标签内容保存起来，保存到 ServiceBean 里面
4，serviceBean 还实现了 ApplicationListener<ContextRefreshedEvent>接口，叫应用的监听器。它监听的事件是 ContextRefreshedEvent，当我们 ioc 容器整个刷新完成，也就是 ioc 容器里面所有对象都创建完成以后来回调方法 onApplicationEvent(ContextRefreshedEvent event。
5，监听方法主要是暴露方法的 URL 地址
6，接下来分析 protocol，得到端口号
7，拿到这个 URL 之后开启 nettyServer 服务器，并且监听之前得到的端口号
8，DubboExporter 来开启 netty 服务器，registryExporter 用来注册，服务（执行器）和对应的 url 地址，注册到注册表里。

1.22 如果秒杀服务运行过程中 mq 服务挂掉了怎么办?

1，首先要从设计角度，保证 mq 的高可用性，做集群和主从备份
2，需要给 MQ 配置消息持久化，防止挂掉了之后消息丢失
3，做好监控，防止服务挂了之后没有检测到，并且及时把服务重启起来
4，从业务角度来保证 MQ 挂了之后用户的体验

1.23 mq 消费者如果消费不成功怎么办呢?

1，MQ 如何消费不成功，RocketMQ 会把该消息丢入重试队列当中去
2，MQ 重试队列默认有重试机制，会反复去投递直到投递成功或者是在 16 次（默认的，可

更改）之后假如还没有消费失败的话会把该消息投递到 MQ 死信队列里面
3，我们可以通过后台管理系统从死信队列里面把没消费成功的消息取出来，然后手动的去进行库存的扣减

1.24 说一下熔断降级和限流?

服务的熔断与降级：

服务熔断：熔断这一概念来源于电子工程中的断路器（Circuit Breaker）。在互联网系统中，当下游服务因访问压力过大而响应变慢或失败，上游服务为了保护系统整体的可用性，可以暂时切断对下游服务的调用。这种牺牲局部，保全整体的措施就叫做熔断。

服务降级：服务降级是从整个系统的负荷情况出发和考虑的，对某些负荷会比较高的情况，为了预防某些功能（业务场景）出现负荷过载或者响应慢的情况，在其内部暂时舍弃对一些非核心的接口和数据的请求，而直接返回一个提前准备好的 fallback（退路）错误处理信息。这样，虽然提供的是一个有损的服务，但却保证了整个系统的稳定性和可用性。

熔断 vs 降级：

相同点：

目标一致，都是从系统的可用性和可靠性出发，为了防止系统崩溃

用户体验类似，最终都让用户体验到的是某些功能的不可用

不同点：

触发原因不同：服务熔断一般是某个服务（下游服务）故障引起，而服务降级一般是从整体负荷考虑

服务熔断与降级实现框架：Hystrix

服务限流：

见面试题目 1.14

1.25 项目如何登陆?

采用 redis 单点登录，见 1.15

1.26 说一下消息队列消息的类型有哪些?

普通消息：普通消息也叫做无序消息，简单来说就是没有顺序的消息。因为不需要保证消息的顺序，所以消息可以大规模的并发的发生和消费，吞吐量很高，适合大部分场景

有序消息：有序消息就是按照一定的先后顺序的消息类型。
有序消息还可以进一步分为：全局有序消息（1 个 MessageQueue）、局部有序消息（多个

MessageQueue）

延时消息：简单来说就是当 producer 将消息发送到 broker 之后，会延时一定时间后才投递给 consumer 进行消费。

事务消息：本地事务和事务消息的投递保持一致性

1.27 项目的 TPS 和 QPS 是多少?

开放性问题，回答合理即可，一般来说，调优后的机器单台 TPS 可到达 1500，QPS 在 2000，当然这个数据也会和物理机的配置有关。

假如是单机 2000，那么配置 8 个基点的 QPS=2000 * 8 = 16000
回答个大概数字（万级，十万级，百万级）即可，上线之后节点数量可以增减，所以可以动态的调整这个数字