抢购流程梳理

注册中心服务（config-server）：

application.yml配置中认证方式，使用ssh认证

1.git中创建配置仓库（shop-cloud-config），私有化

2.创建完后新增zuul-server.yml配置文件并添加内容

3.克隆/下载（复制SSH地址），粘贴到config-server的application.yml配置中的uri

4.生成公钥和私钥，任意目录右键Git Bash Here，输入ssh-keygen.exe -t rsa -m PEM -C "yichengc3@163.com"

5.生成后存放路径，本地磁盘->用户->.ssh，.ssh文件夹中id\_rsa是私钥id\_rsa.pub是公钥

6.添加以下两个配置：

ignore-local-ssh-settings: true

private-key: |

-----BEGIN RSA PRIVATE KEY-----

<第5步骤私钥>

-----END RSA PRIVATE KEY-----

7.将公钥添加到仓库（shop-cloud-config）中，用户栏->设置->安全设置->SSH公钥，第5步骤公钥粘贴到仓库公钥中

网关服务（zuul-server）：

问题：登录完成后，参与抢购活动等待活动的开始，半个小时过去此时当活动开始了，此时操作需要登录并跳转到登录页面，不太友好

解决：用户只要在系统进行相应的操作或者刷新页面，就表示该用户是存活着，（刷新token）

1.所有进入服务的请求都会先经过zuul网关

2.用户请求到任意服务，都需要刷新redis和cookie中token的有效时间

3.利用ZuulFilter来做token刷新处理

4.利用ZuulFilter的后置拦截，完成token刷新

4.1.校验请求中是否携带cookie(token)

4.2.如果有携带token，则刷新redis和cookie的有效时间

LoginTokenFilter

会员服务（member-server）：

接口:

SpringMVC中的@CookieValue注解的运用，（@CookieValue(name = Constants.USER\_LOGIN\_TOKEN) String token）

用户参数解析器的运用，（@UserParam User user），通过对象的注入方式直接获取用户并返回，只有贴上@UserParam注解才注入到bean，减少请求调用service层，需配置UserArgumentResolver类解析器和MemberWebConfig配置类（将解析器注入到bean中），通过启动配置直接调用

秒杀服务（seckill-server）：

问题：

1.秒级别的并发量出现重复下单情况  
2.秒级别的并发量出现库存负数情况

解决：

1.1.t\_seckill\_order表中user\_id字段和seckill\_id字段建立唯一索引且方法上贴上@Transactional(rollbackFor = Exception.class)  
 （但程序执行过程中会出现->Duplicate entry 'user\_id的值-seckill\_id的值' for key 'user\_id'）会有串行执行

建立唯一索引（UNIQUE）：ALTER TABLE `t\_seckill\_order` ADD UNIQUE (`user\_id`,`seckill\_id`)  
 备注：存在性能问题

1.2.通过redis单线程特性，方法setIfAbsent/increment(自增，例如每个用户只下单3个商品之后得出的结果与3判断)实现重复下单校验  
 此校验只能作为预判断，如果其它业务逻辑处理失败，将标识删除

拓展：redis数据安全性问题，数据丢失；通过持久化机制（rdb/aof）处理，4.0新版本推出的混合持久化，也就是利用rdb/aof的特性组成的持久化；需要做redis集群，因为会存在单机故障的问题，需要异地多活（跨机房部署），其中一个机房可能会因X因素，使得整个机房无法访问，集群+跨机房部署确保redis数据不丢失

2.1.通过乐观锁思想，修改时sql语句用库存大于0作为条件，大于0减1操作，否则不操作  
 （但请求上千甚至上万，都会执行秒杀中前置的所有判断，包括减库存判断，减库存判断是操作数据库）  
 备注：存在性能问题  
 2.2.通过redis将秒杀商品的库存缓存，程序执行过程中操作redis中的缓存进行库存预减，方法increment  
 （但请求上千甚至上万，还是会有请求到redis）

备注：redis压力过大  
 2.3.通过二级缓存（本地缓存，缓存到JVM中），定义一个map标识作为库存是否售完，得出结果来拦截绝大部分不可能秒杀成功的请求

为什么HashMap是线程不安全？

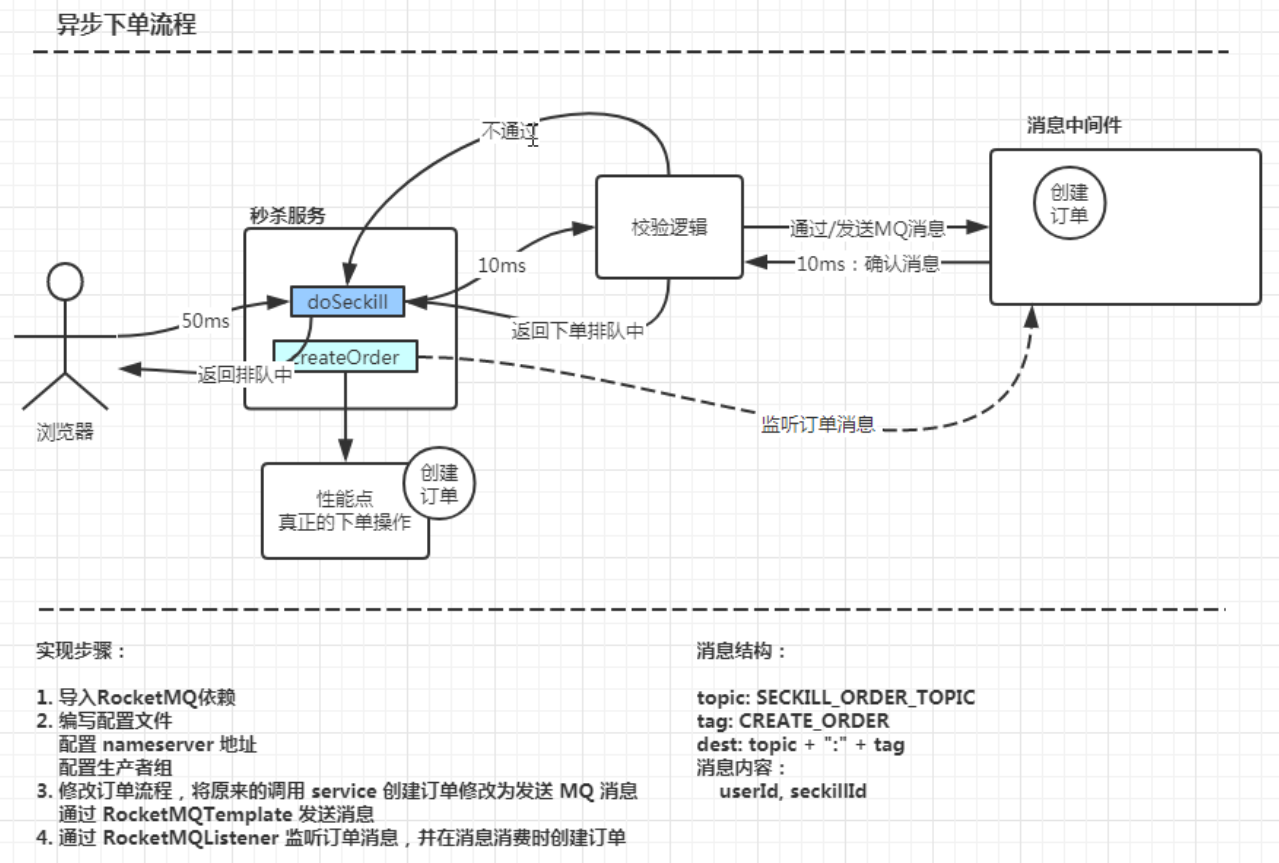
put后的key会经过hash算法，得到一个值，这个值作为存储在数组中的位置，在计算hash值的时候，有可能会出现hash冲突

为什么HashTable、ConcurrentHashMap是线程安全？

HashTable用了synchorinzed锁机制，在方法上上锁，只有一个线程能访问，其他线程等待

ConcurrentHashMap用了分段锁，在分段空间中进行上锁，可多线程访问

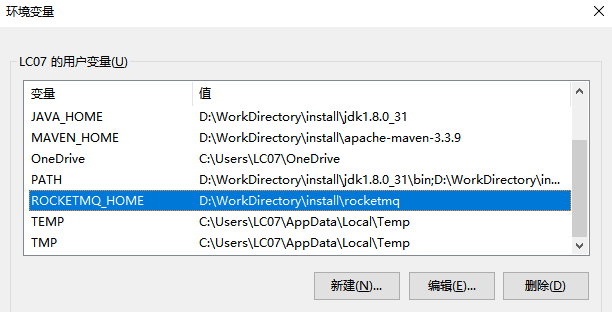
MQ服务（异步下单）：



1.RocketMQ环境准备

下载：<http://rocketmq.apache.org/release_notes/release-notes-4.9.0/，解压>

配置环境变量



启动：

runbroker.cmd与runserver.cmd，配置中set "JAVA\_OPT=%JAVA\_OPT% -server -Xms2g -Xmx2g -Xmn1g"

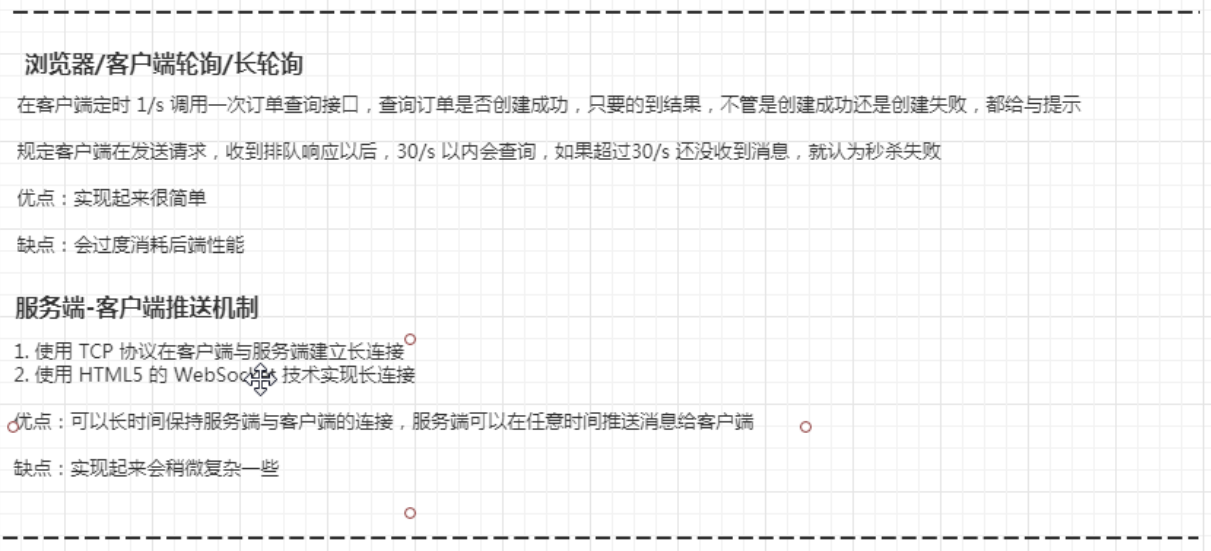
可修改为：set "JAVA\_OPT=%JAVA\_OPT% -server -Xms512m -Xmx512m -Xmn256m"

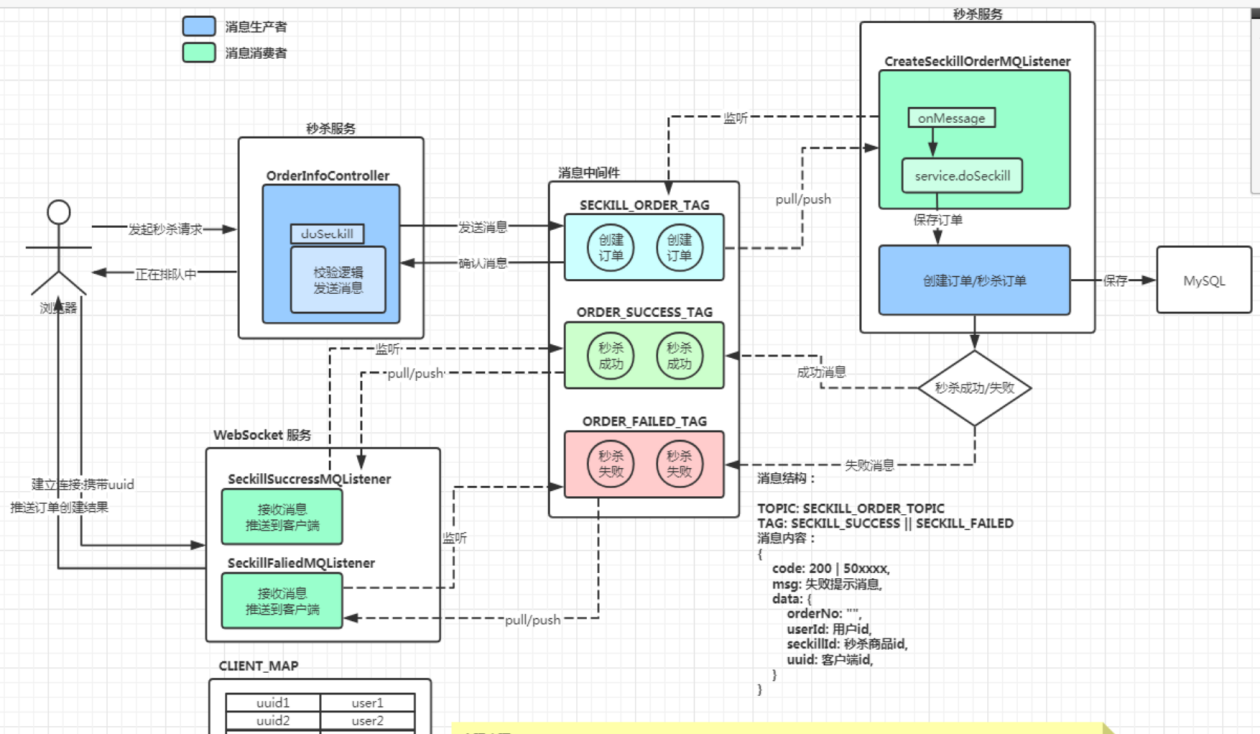
Xms：初始化内存、Xmx：最大内存、Xmn：年轻代内存

双击mqnamesrv.cmd，用命令启动mqbroker.cmd->mqbroker -n localhost:9876 autoCreateTopicEnable=true

代码实现消息发送

2.正因为异步下单，当用户下单时，不通过前置校验直接返回对应异常信息，通过前置校验返回正在下单中，此时如何将最终下单成功或失败的结果通知给用户（使用服务端与客户端的通信机制，基于WebSocket的长连接推送机制）





3.用户下单后返回的是正在下单中，此时就需要发起一个连接到WebSocket服务，这个服务是专门和客户端进行通信的服务，且发起连接携带用户的唯一标识，会将客户端的连接保存到Map中，key表示用户的唯一标识，value表示用户的连接对象

4.搭建WebSocket服务

5.点击秒杀

问题：

1.点击秒杀按钮，因为创建订单和建立websocket连接是异步进行，会出现订单未创建，建立连接中获取客户端，获取为空

2.进入秒杀界面就建立连接，此时websocket服务压力增大，频繁的建立，断开操作

解决：

使用重试机制

6.订单生成

问题：

订单在规定时间范围内未支付，通过RocketMQ发送延迟消息,检查订单是否超时未支付

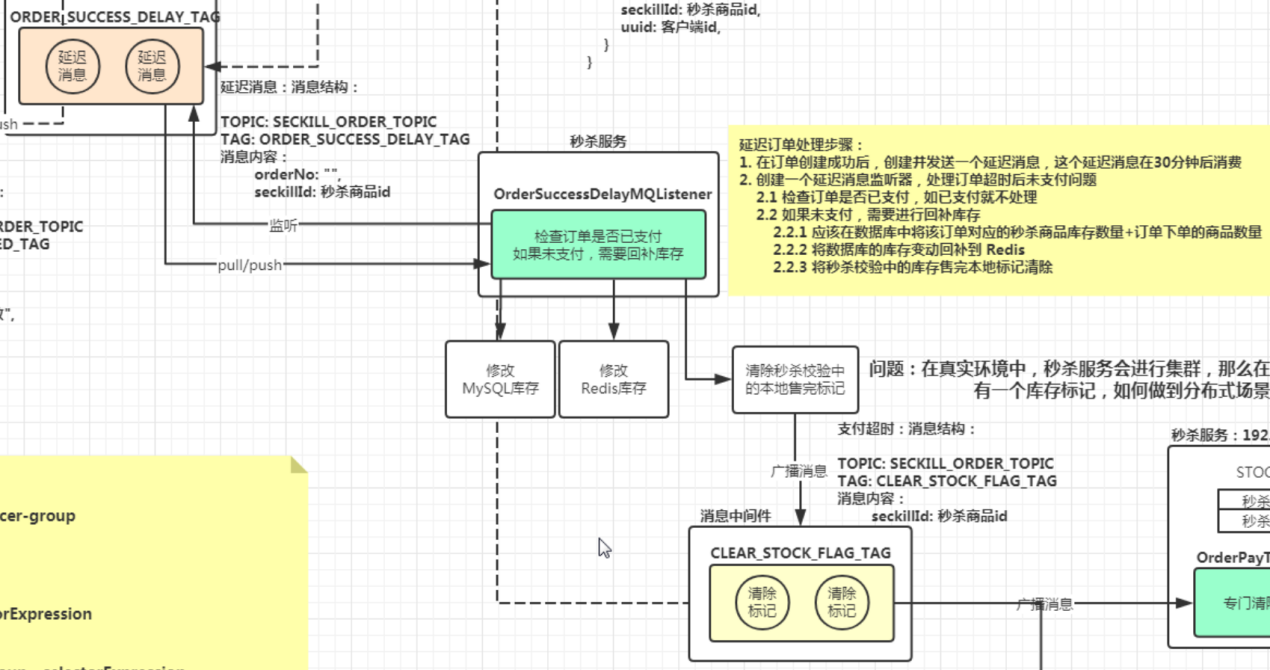
解决：

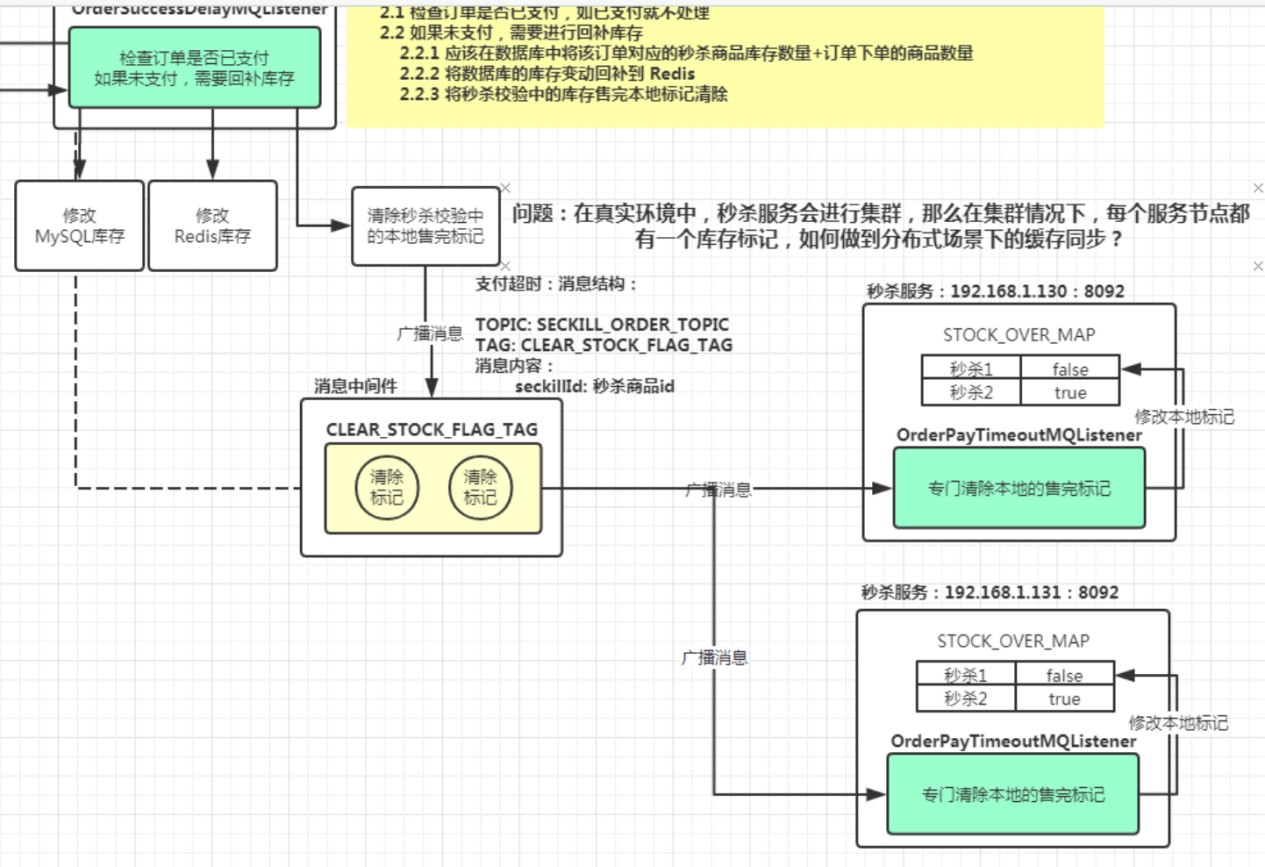
订单创建成功后就要发送延迟消息，当订单一直未支付，由消费者一直监听着这条消息，时间一到由消费者消费这条消息。检查订单状

态，如果未支付，则修改订单状态；如果未支付，回补数据库库存；数据库库存同步到redis中

重点：

本地售完标识的处理，发送广播消息，清除本地售完标识（可多个消费者消费，集群秒杀服务，多个实例，解决分布式缓存同步问题）





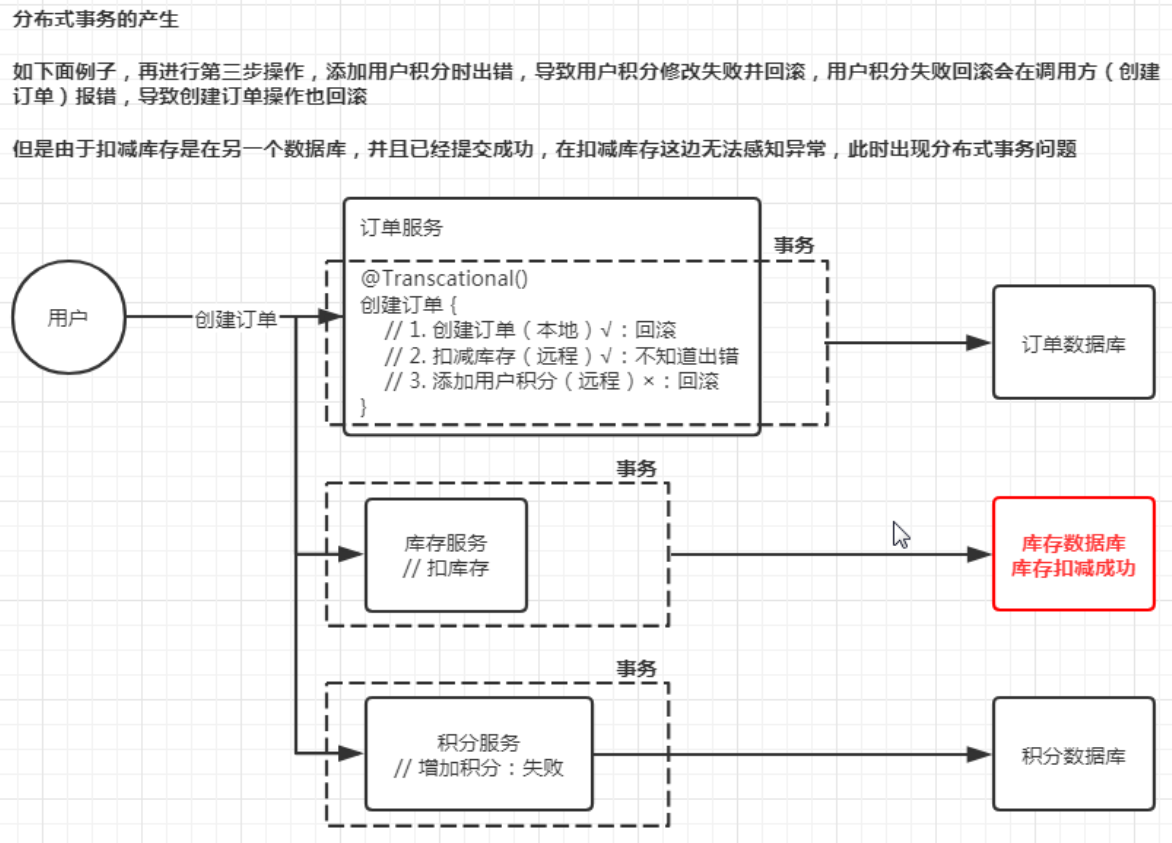
7.订单支付宝支付

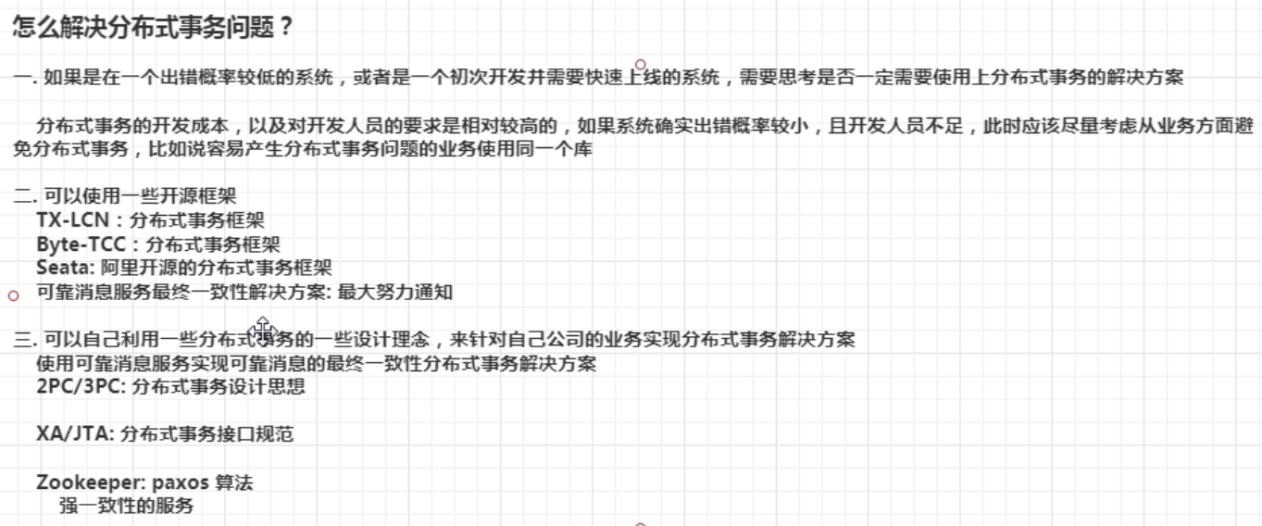
支付订单异步回调业务处理失败补偿方案

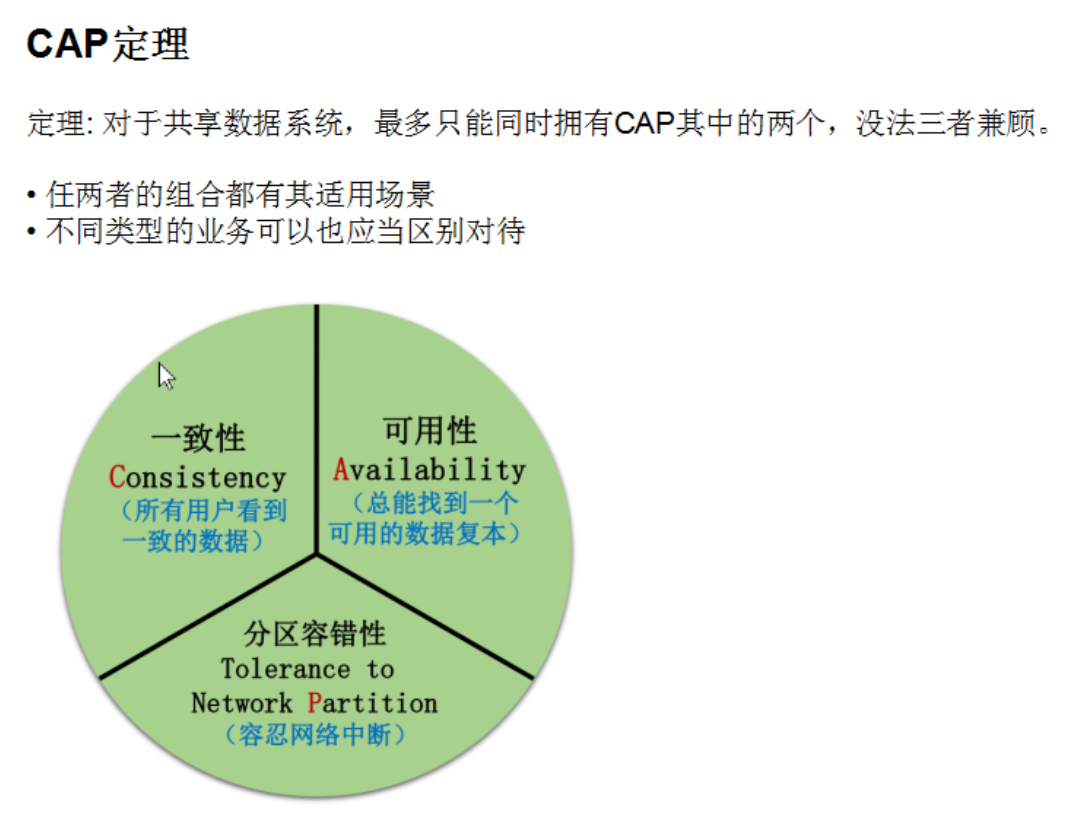
解决：

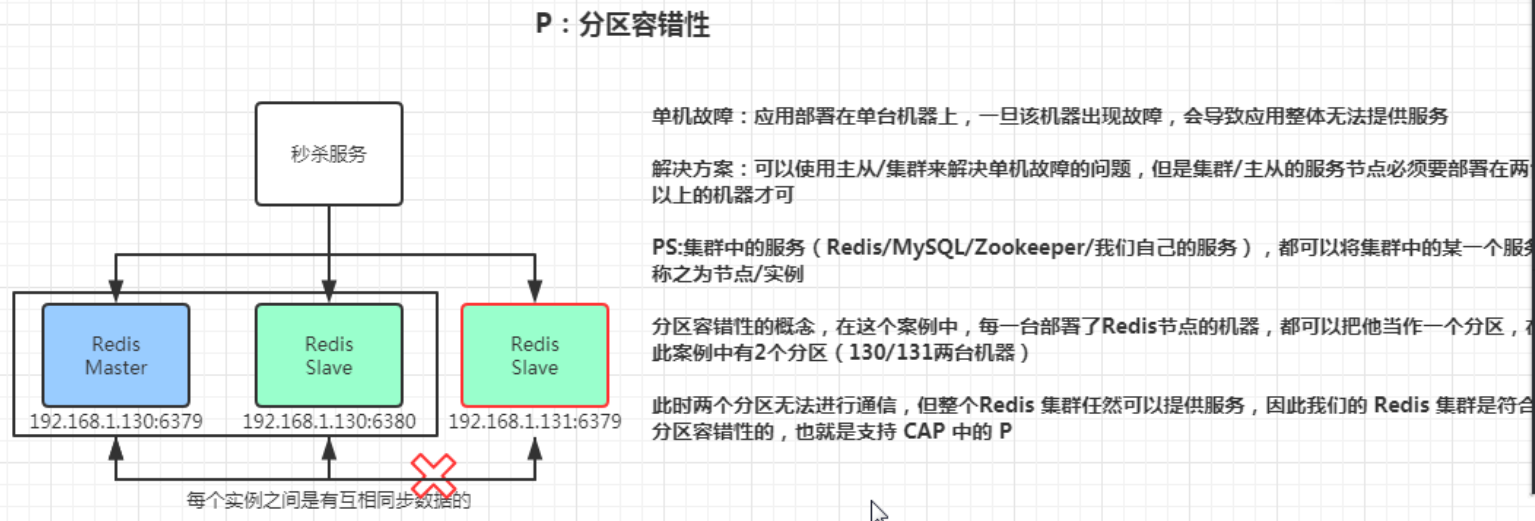
订单创建成功后发送延迟消息，检查订单是否超时未支付，由消费者一直监听着这条消息，时间一到由消费者消费这条消息，在方法中先通过支付宝查询订单接口，查询该订单是否未支付，如果未支付把本地订单状态修改为已超时，如果已支付表示代码业务中出问题，把本地订单状态修改为已支付

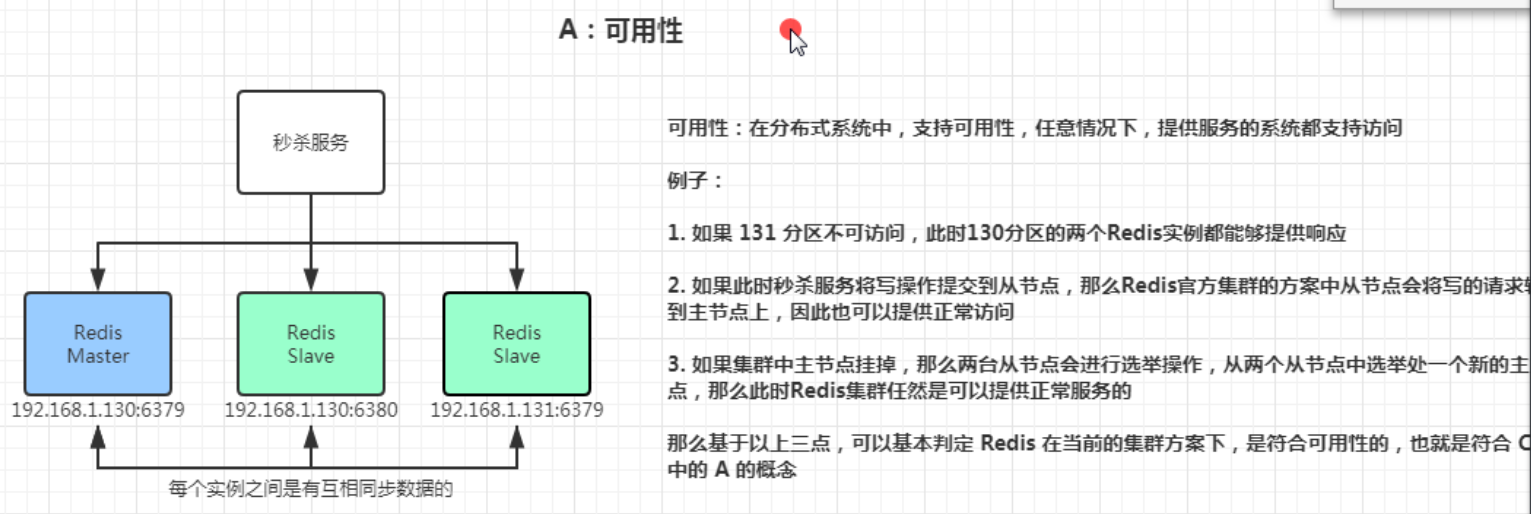
分布式事务：

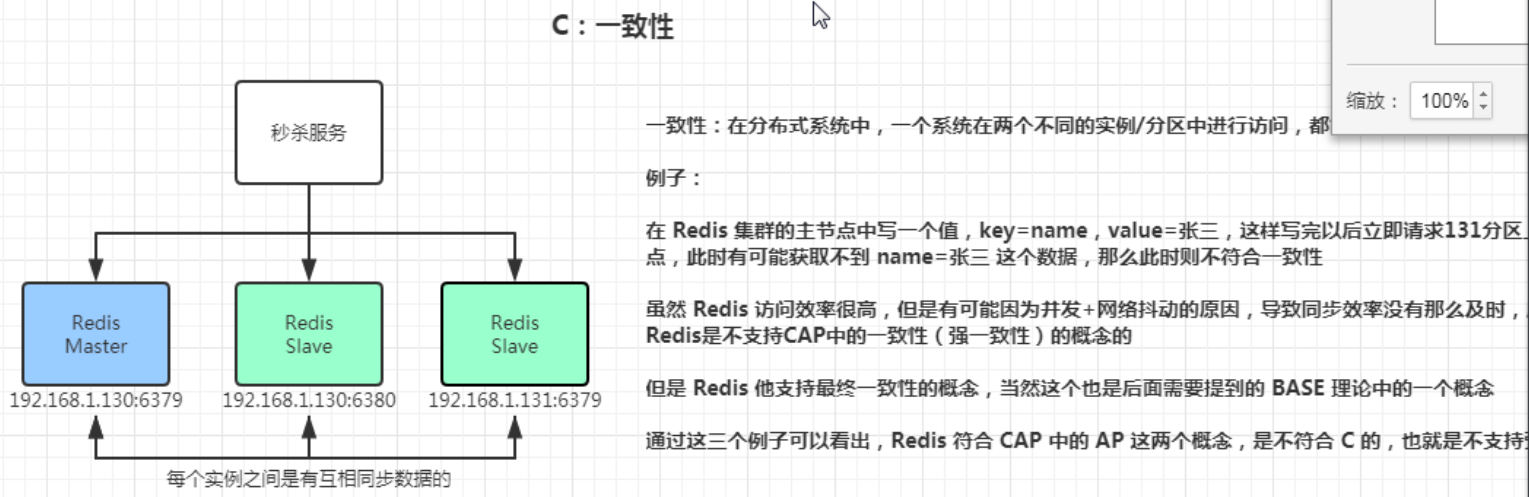






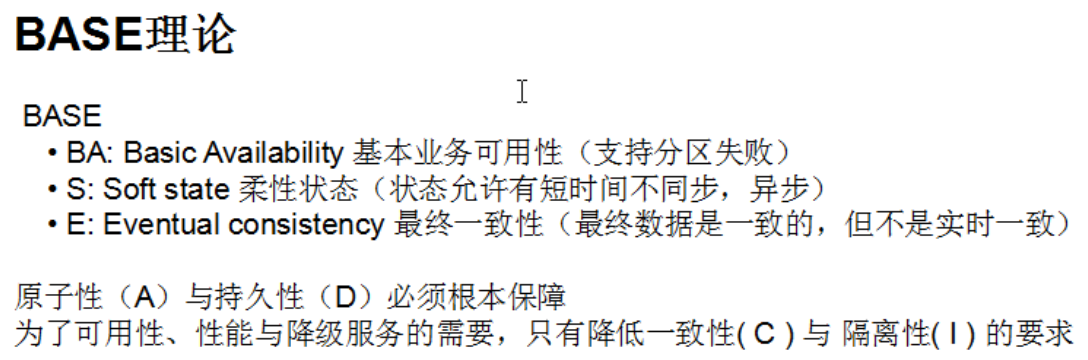








CAP定理带来的问题由BASE理论进行补偿：

可靠消息最终一致性处理：

正常流程：

1.生产者在执行业务操作前调用消息服务子系统持久化预发送消息记录，该记录状态为预发送

2.执行业务操作并持久化

3.生产者将业务操作发送，调用消息服务子系统确认消息并往MQ发送消息，前提是执行业务操作成功，那么该预发送消息记录此时状态修改为可消费，并发送MQ消息到消息中间件，消费者则监听到该消息并消费且持久化，之后调用消息服务子系统确认消息已被消费，最后将该可消费记录状态修改为已消费

异常流程：

1.生产者在执行业务操作前调用消息服务子系统持久化预发送消息记录，失败，不影响系统

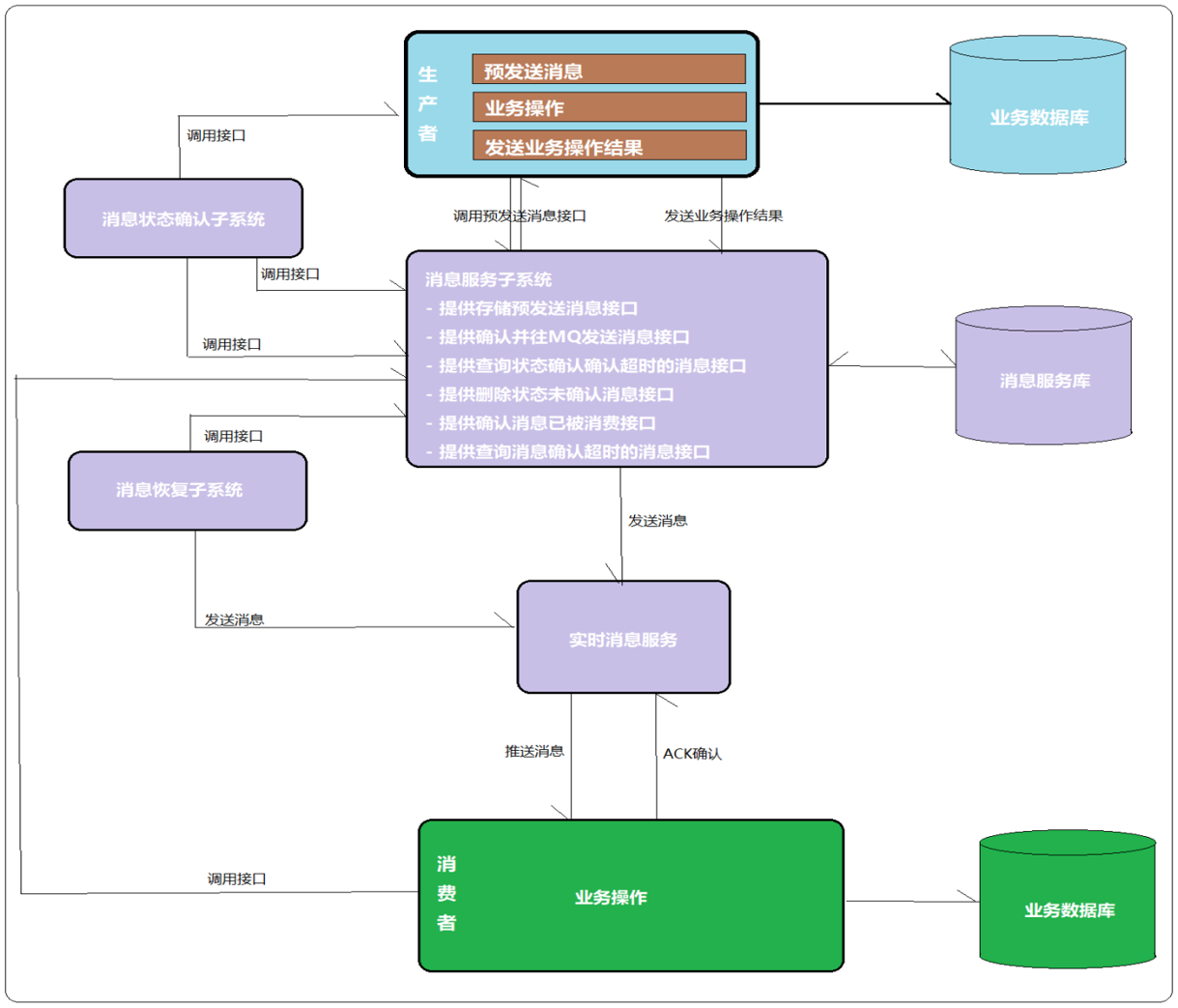
2.执行业务操作并持久化，失败，但是消息服务子系统数据库已存储预发送记录，如何处理该记录，会有个消息状态确认子系统，通过定时任务调用消息服务子系统查询状态确认确认超时的消息（预发送的消息），之后消息状态确认子系统在调用生产者服务业务操作结果确认查询数据库记录是否变更，已变更则消息服务子系统数据库记录状态修改为可消费，未变更则删除

3.业务操作成功，生产者将业务操作发送，失败但不会做回滚，同样执行消息服务子系统后面的逻辑，发送MQ消息，此时发送失败，如何处理，会有个消息恢复子系统，通过定时任务调用消息服务子系统查询状态确认确认超时的消息（可消费的消息），将消息取出进行重发到消息中间件，消费者则监听到该消息并消费且持久化，之后调用消息服务子系统确认消息已被消费，最后将该可消费记录状态修改为已消费

优点：消息系统独立，解耦得到优化，服务压力得到缓解，持久化机制可用到不同方式存储，例如redis，mongodb等等

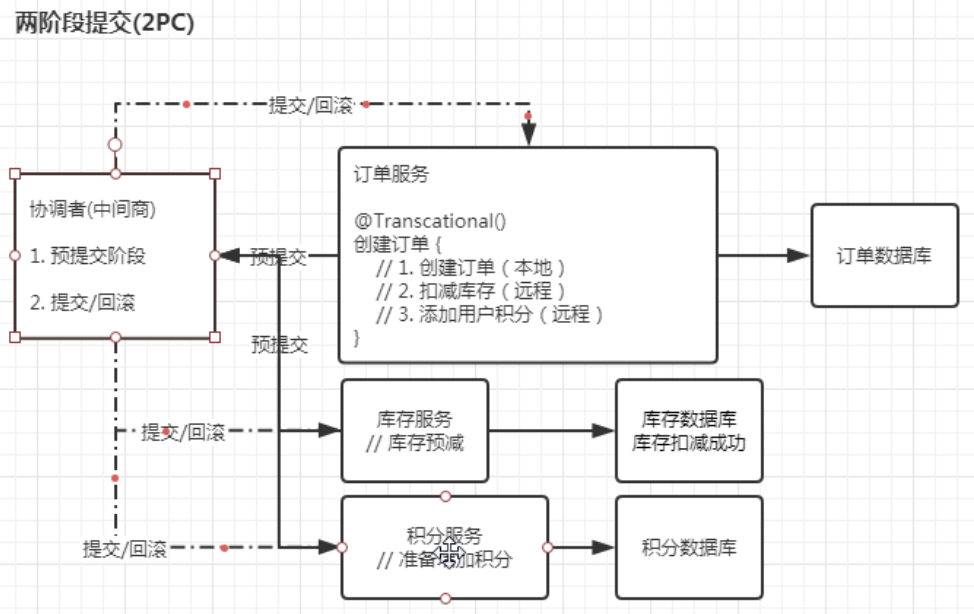
缺点：生产者一旦创建业务，那么消费者必须消费，但不一定所有消费者都会消费成功，例如订单创建成功，但用户余额不足，无论如何重

试任然得不到解决；消费者存在消费失败的业务可靠消息最终一致性的方案不适用，也就不支持回滚功能

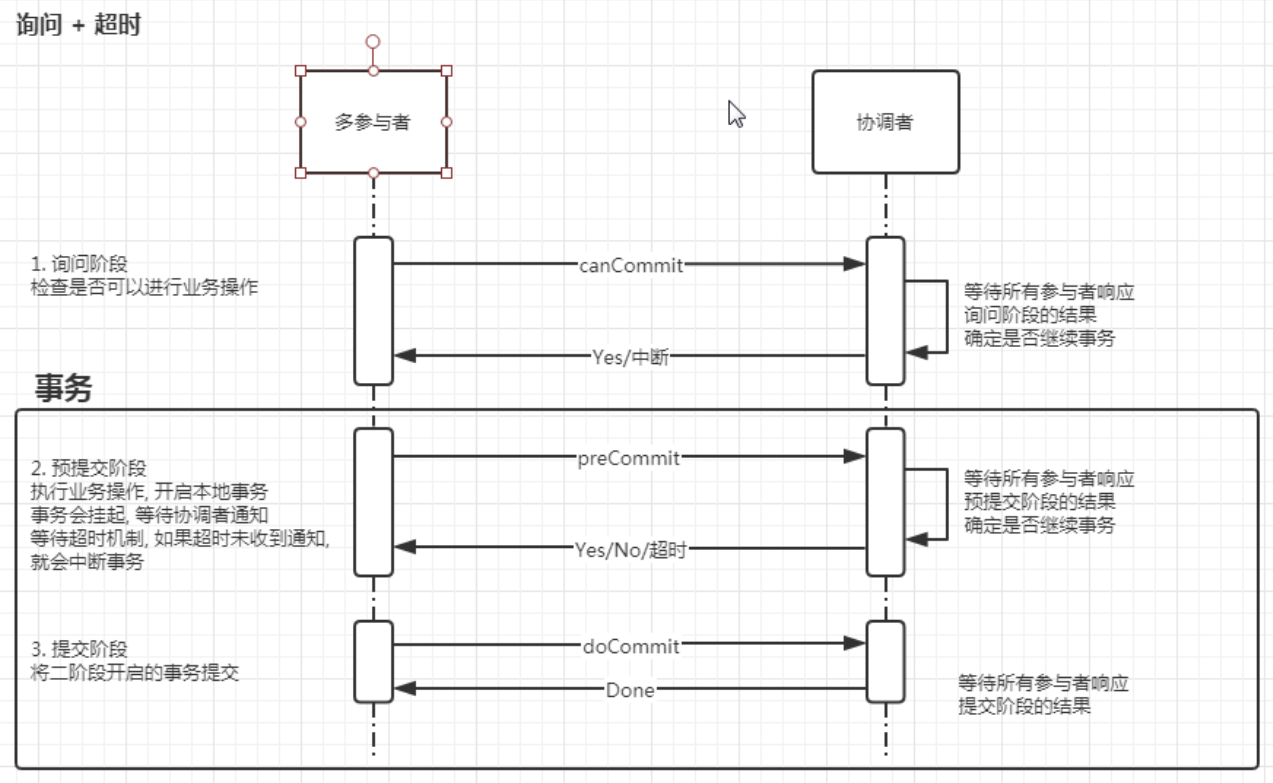


两阶段提交：把事务提交分为几个阶段，将订单服务、库存服务、积分服务执行的业务预提交到事务管理器服务，只要其中一个参与方服务执行业务出现异常，其余参与方服务都会进行回滚，否则提交，事务管理器服务起到一个统一管理事务的角色

问题：参与方服务都执行成功，当其中一个参与方服务超时，导致整体业务操作出现数据缺失

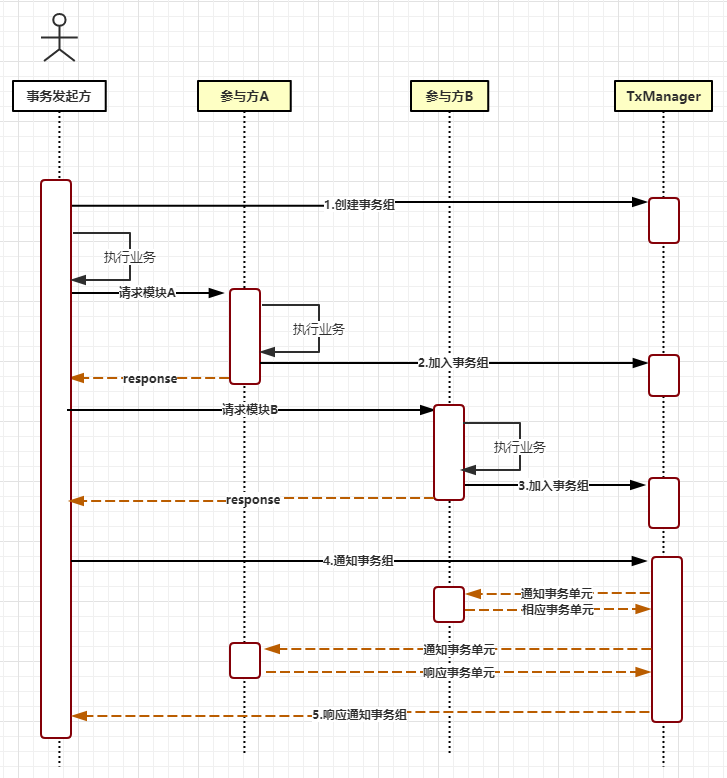


三阶段提交：同样把事务提交分为几个阶段，但前提提交前会先通过询问阶段，等待所有参与方服务响应询问阶段的结果，确认是否继续事务，继续事务则执行两阶段的操作，否则不继续（杜绝了一部分资源不足的情况），且从中有超时机制的处理，当参与方服务都执行成功，其中一个参与方服务超时，此时会将其他已成功的事务变成回滚，从而达到整体回滚



解决分布式事务问题（TX-LCN框架）

实现思路图：



解析：TxManager事务管理器服务（协调者）需启动，事务发起方开启事务，通过访问事务管理器服务创建事务组，继续执行自身业务

在业务中调用远程服务参与方A服务，A服务执行自身业务后加入到事务组并将结果（成功/失败）响应给发起方

在业务中调用远程服务参与方B服务，B服务执行自身业务后加入到事务组并将结果（成功/失败）响应给发起方

A服务、B服务事务并没有真正提交，而是被事务管理器服务代理

业务执行完后发起方通知事务组，此时有事务在执行，事务管理器服务通过上述响应结果通知事务单元，以哪个是最后一个提交事务组顺序来进行通知提交/回滚

例如B服务是最后一个参与方，通知B服务事务提交/回滚，之后在通知事务管理器服务提交/回滚，以此类推最终到发起方提交/回滚

实现流程：

1.下载https://github.com/codingapi/tx-lcn，解压

2.txlcn-tm事务管理器入口，修改pom文件内容加入打包插件

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

</executions>

</plugin>

3.cd到项目resources目录下，执行tx-manager.sql创建该表，注意修改create\_time类型datetime(0)

4.修改数据库连接信息，包括url加上时区，密码

5.打包mvn clean package -Dmaven.test.skip=true

6.打包完后运行target中的jar，跑起项目

7.集成到微服务项目中，添加依赖（事务的发起方，参与方）

<dependency>

<groupId>com.codingapi.txlcn</groupId>

<artifactId>txlcn-tc</artifactId>

<version>5.0.2.RELEASE</version>

</dependency>

<dependency>

<groupId>com.codingapi.txlcn</groupId>

<artifactId>txlcn-txmsg-netty</artifactId>

<version>5.0.2.RELEASE</version>

</dependency>

8.启动类中贴注解@EnableDistributedTransaction，启用分布式事务

9.业务方法贴注解@LcnTransaction，分布式事务注解

10.运行微服务项目出现异常，request fail. non tx-manager is alive，重新启动事务管理器服务