自我介绍：

我叫\*\*，来自\*\*，\*\*\*\*部，交易组，功能：支付、查询流程，对Java基本数据结构，熟悉常用框架，了解RCPdubbo，缓存，jvm gc了解。

我叫\*\*，来自\*\*，\*\*\*\*部，交易组，业务流程代码开发，业务校验、入库、发送银行、接收银行，更新库，返回机构，多种交易类型。还要异步流程，延迟队列。服务降级，发送给银行，延迟发送，白名单。缓存。基础信息，定时从缓存获取。银行机构交互用了dubbo。

工作努力，勤学好问，乐观，健身，瑜伽。

问题一：你是如何理解长连接的？有哪些方式实现长连接？

回答：\*\*跟机构是长连接，交易跟其他上下游是长连接，跟数据库不确定。http不确定。

答案：

长连接：建立TCP连接后，通过发送心跳包保活，只有在异常情况下主动关闭连接，否则一直使用。长连接本质上是在传输层实现的。

实现方式：

1. 使用TCP内嵌心跳机制，可以在客户端或服务端配置socket参数SO\_KEEPALIVE为true。保活默认是关闭的。同时配合3个Linux参数，分别是：

tcp\_keepalive\_time：打开保活情况下，连接允许的最大空闲时间，默认2小时

tcp\_keepalive\_intvl：当空闲时间超过tcp\_keepalive\_time发送心跳探测包的频率，默认75s

tcp\_keepalive\_probes：心跳探测包发送次数，默认9次

参数位于：/proc/sys/net/ipv4/{参数}

2. 协议层面，某些应用层协议本身支持长连接，例如HTTP 1.1，本质还是在TCP上配置了长连接参数

3. 应用层主动发送心跳，可以发送自定义协议的心跳包，携带一些必要的参数，这种心跳包不仅有保活的作用，还有业务含义。例如当前同步数据偏移量，集群状态，路由表等。当应用系统需要自定义心跳检测时通常会关闭TCP保活，因为应用心跳包的发送频率通常会大于TCP心跳频率。应用心跳往往无需应答，因此是Ping-Ping式，而非传统的Ping-Pong式。

问题二：共有2n个数，寻找其中最大值和最小值最少的比较次数？

回答：先算最大值，再算最小值，2n – 1 + 2n -2 = 4n – 3。

答案：先找出两个数比较一次，算出最大值、最小值，剩下2n -2个数每次取两个值比较，找出一大一小，大的跟最大值比较，小的跟最小值比较，因此比较3次，剩余2n -2个数需要(2n - 2) / 2 \* 3次，最终为3n -2次。

问题的关键是两两比较中较大的数，一定比最小值大，所以无需比较，只需跟最大值比较。

问题三：如何减少锁的竞争

回答：使用读写锁；减少锁的粒度，只加在有并发修改的地方

答案：有两个因素影响锁的竞争性：锁的请求频率和持有锁的时间。如果两者乘积较小，则竞争性小。

方法一：减少同步代码块中不必要的代码，降低锁持有的时间。在同步代码块中只包含必要的原子操作。

方法二：使用锁分解，降低锁粒度。如果一个锁保护多个变量，可以用多个锁分别保护一个变量，原来修改不同变量需要竞争同一个锁变为竞争多个锁，只有对同一变量的并发操作才需要竞争，降低锁请求频率。同理可以使用分段锁，一段数据可以用多个锁保护，每个锁保护一段数据。ConcurrentHashMap实现了16个分段锁保护一个数组，因此可以支持更高的并发，如果散列访问，并发是独占锁的16倍。但分段锁的缺陷是如果要对数据进行独占访问时需要获得多个锁。锁分解和锁分段都是让不同线程在不同数据上可以并发操作。

方法三：避免热点域，跟解决热点账户的思路类似，原来有一个热点账户，有大量的操作修改同一个值，可以将一个大账户分解为多个小账户，操作多个子账户。大账户的总额是所有小账户的金额的累加。但一个问题是总额可能不是最准确的，因为在对小账户的金额进行累加时，累加过的小账户金额可能已经修改了，所以导致不准确。降低锁的请求频率。避免热点域是分段锁的一种应用

方法四：使用并发容器例如ConcurrentHashMap、读写锁、原子变量代替独占锁

方法五：使用无锁设计

总结：

1. 优化代码，精简同步代码，不要用一个锁管理多个独立对象

2. 优化代码，用并发容器、读写锁、原子变量代替独占锁，提高并发性

3. 拆分锁，把一个大对象由一个锁保护，拆成多个小对象，分别由多个锁保护，提高并发性