Question2

@Transactional

> 每个带有@Transactional注解的方法都会创建一个切面,所有的事务处理逻辑就是由这个切面完成的,这个切面的具体实现就是TransactionInterceptor类.
>
> 注意,这个TransactionInterceptor是个单例对象,所有带有@Transactional注解的方法都会经由此对象代理(省略无关代码):
>
> @Transactional 应用在非 public 修饰的方法会失效
>
> 原因是TransactionInterceptor （事务拦截器）在目标方法执行前后进行拦截的时候会去对修饰符校验是否为public，不是public不会生效
>
> 同一个类中方法调用，导致@Transactional失效
>
> 由于使用Spring AOP代理造成的，因为只有当事务方法被当前类以外的代码调用时，才会由Spring生成的代理对象来管理。内部方法的调用不会使用代理，也就无法解析@Transactional

Codis与redis cluster

> codis通过zookeeper或者etcd等配置中心保存元数据信息，实现不同codis实例之间的数据同步
>
> codis扩容
>
> Codis 对 Redis 进行了改造，增加了 SLOTSSCAN 指令，可以遍历指定 slot 下所有的key。Codis 通过 SLOTSSCAN 扫描出待迁移槽位的所有的 key，然后挨个迁移每个 key 到新的 Redis 节点。在迁移过程中，Codis 还是会接收到新的请求打在当前正在迁移的槽位上，因为当前槽位的数据同时存在于新旧两个槽位中，Codis 如何判断该将请求转发到后面的哪个具体实例呢？Codis 无法判定迁移过程中的 key 究竟在哪个实例中，所以它采用了另一种完全不同的思路。当 Codis 接收到位于正在迁移槽位中的 key 后，会立即强制对当前的单个 key 进行迁移，迁移完成后，再将请求转发到新的 Redis 实例。

redis事务与序列化

> Redis 事务可以一次执行多个命令， 并且带有以下三个重要的保证：
>
> - 批量操作在发送 EXEC 命令前被放入队列缓存。
> - 收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行。
> - 在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。
>
> 一个事务从开始到执行会经历以下三个阶段：
>
> - 开始事务。
> - 命令入队。
> - 执行事务。
>
> redis事务不保证原子性，更像是一种批量/打包执行的命令脚本。
>
> ```
> multi命令代表事务开始， exec命令代表事务结束，它们之间的命令是原子顺序执行的。
> ```
>
> redis不支持回滚

ThreadLocal 内存泄漏的原因

threadLocalMap使用ThreadLocal的弱引用作为key，如果一个ThreadLocal不存在外部**强引用**时，Key(ThreadLocal)势必会被GC回收，这样就会导致ThreadLocalMap中key为null， 而value还存在着强引用，只有thead线程退出以后,value的强引用链条才会断掉。

每次使用完ThreadLocal都调用它的remove()方法清除数据

场景1：

ThreadLocal 用作保存每个线程独享的对象，为每个线程都创建一个副本，这样每个线程都可以修改自己所拥有的副本, 而不会影响其他线程的副本，确保了线程安全。比如数据库连接对象

场景2：

ThreadLocal 用作每个线程内需要独立保存信息，以便供其他方法更方便地获取该信息的场景。每个线程获取到的信息可能都是不一样的，前面执行的方法保存了信息后，后续方法可以通过ThreadLocal 直接获取到，避免了传参，类似于全局变量的概念。用 ThreadLocal 保存一些业务内容（用户权限信息、从用户系统获取到的用户名、用户ID 等），这些信息在同一个线程内相同，但是不同的线程使用的业务内容是不相同的。

> 我们知道，一个ThreadLocal实例对应当前线程中的一个TSO实例。因此，如果把ThreadLocal声明为某个类的实例变量（而不是静态变量），那么每创建一个该类的实例就会导致一个新的TSO实例被创建。显然，这些被创建的TSO实例是同一个类的实例。于是，同一个线程可能会访问到同一个TSO（指类）的不同实例，这即便不会导致错误，也会导致浪费（重复创建等同的对象）！因此，一般我们将ThreadLocal使用static修饰即可。例如，
>

HashMap扩容时机

所以在JDK1.8源码中，执行树形化之前，会先检查数组长度，如果长度小于64，则对数组进行扩容，而不是进行树形化。

所以发生扩容的时候有两种情况，一种是元素达到阀值了，一种是HashMap准备树形化但又发现数组太短，这两种情况均可能发生扩容。