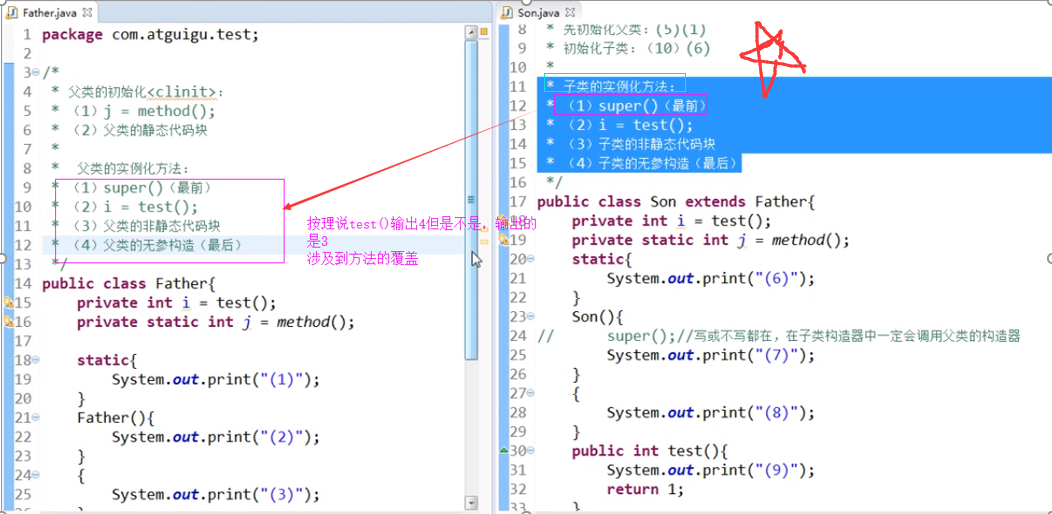
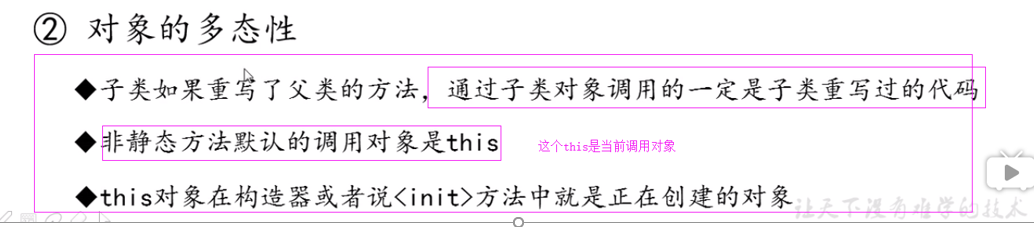
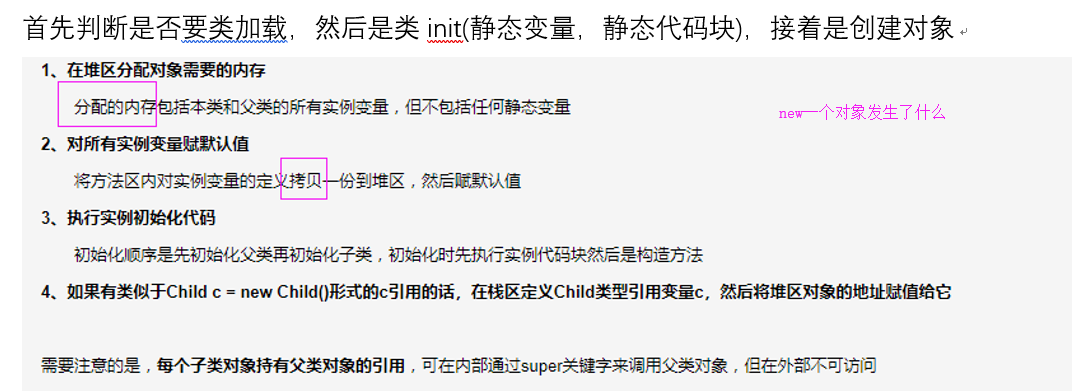
1. int i=1; i=i++先定义i=1的局部变量表，然后把i的值压入操作数栈，然后把局部变量表中的i加1操作，最后把操作数栈中的i赋值给i
2. 

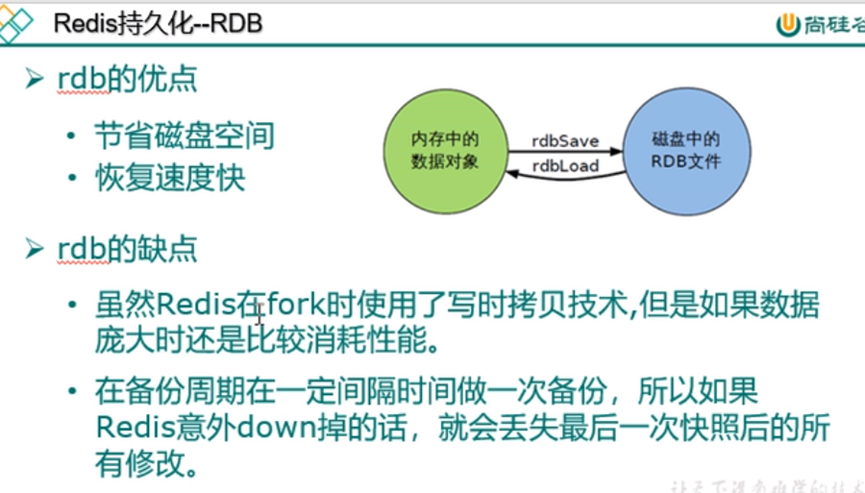


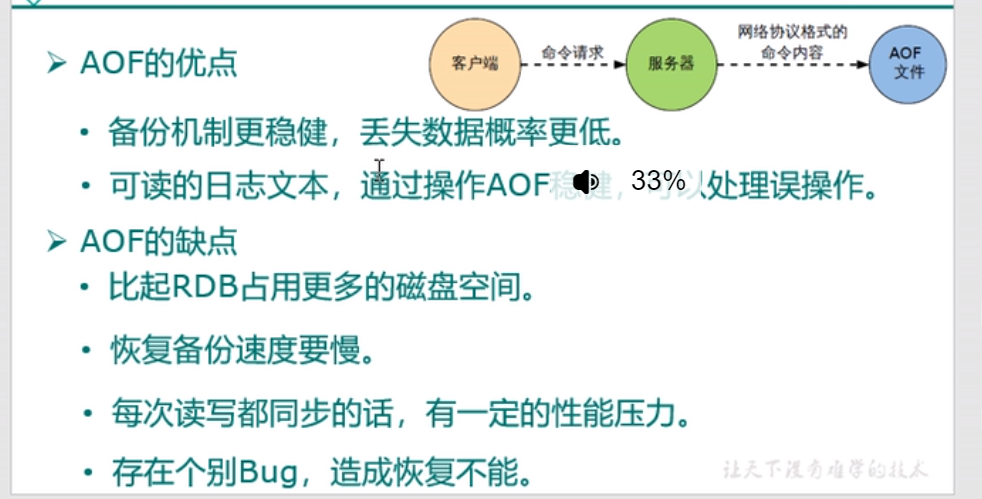
1. 
2. 
3. 数据库事务把异常吃了

Spring Boot 默认的事务规则是遇到运行异常（RuntimeException）和程序错误（Error）才会回滚。自己手动抛出的不会回滚

那这种怎么解决呢？直接往上抛，给上一层来处理即可，千万不要在事务中把异常自己 ”吃“ 掉。

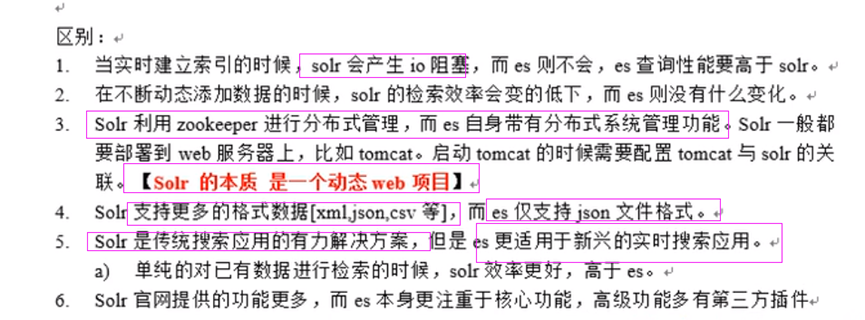
1. 





1. 那些情况需要简历索引（主键，频繁查询的字段，外键，排序，统计，分组）

那些情况不需要简历索引（表记录少，经常增删改，where用不到的，过滤性不好，大文本）

1. redis（String 统计ip次数，防止一个ip多次访问，Hash直接修改某个对象的字段，List消息队列，Set交集，Zset排序）
2. 
3. 单点登录时将token放到cookie中
4. 
5. 
6. 

# 3-4-5部分

7AtomicInteger的底层原理是CAS（调用AtomicInteger的compartAndSet方法，compareAndSet方法调用Unsafe的compareAndSwapInt方法）,里面的变量value用volatile修饰，保证了多线程之间的内存可见性

9 volatile的底层是内存屏障（保证特定操作的执行顺序，保证某些变量的内存可见性）

20collections.synchronizedList实现线程安全的方式是对自己暴力加锁。

21copyonwriteArrayList

22copyonwriteArraySet

23currentHashMap

35

ArrayBlockingQueue

LinkedBlockingQueue 吞吐量比ArrayBlockingQueue高

SynchronousQueue

什么时候唤醒线程什么时候阻塞线程，blockqueue包办了

46线程池的优势：

不再new对象，而是在线程池注入对象

47线程池的3个常用方式

newFixedThreadPool(int nThreads) 就绪线程与最大线程数一致

newSingleThreadExecutor() 单一线程池

newCachedThreadPool() 可缓存的 灵活回收

51线程池的4种拒绝策略（阻塞对列的4种插入）

AbortPolicy(默认直接抛出RejectedExecutionException异常阻止系统正常运行，）

CallerRunsPolicy（调用者运行一种调节机制，该策略即不会抛弃任务，也不会抛出异常，而是将某些人入回退到调用者，从而降低新任务的流量）

DiscardOldestPolicy（抛弃队列中等待最久的任务，然后把当前任务加入队列中尝试再次提交当前任务）

DiscardPolicy（直接丢弃任务，不予任何处理也不抛出异常。如果允许任务丢失，这是最好的一种方案）

55排查出死锁

jsp找出执行的java进程编号，jstack查看这个线程的栈信息

57四大垃圾清楚算法

引用计数法（不用了）

复制

标记清除法（标记清除的垃圾然后回收垃圾会产生碎片）

标记整理（在标清的基础上整理）

58判断对象回收

引用计数法

可达性分析法（根搜索路径）

GCroot对象：栈中引用的对象，方法区静态引用，方法区常量引用，本地方法栈中引用

jdk其他工具使用

jinfo查看某个jvm参数是否开启（查看使用的垃圾收集器，使用的是那种垃圾收集器以及最大堆内存，最大元空间，最大垃圾回收时间）

参数配置一般都是-xx:\*\*这种模式

-Xms -XX:InitialHeapSize

-Xmx -XX:MaxHeapSize

进行jvm调优的时候：java -XX:+PrintFlagsFinal

=是没有被修改

：=是人为修改或者是根据硬件修改

强软弱虚引用

强应用时造成内存泄漏的主要原因

软弱：用一个hashmap来保存图片的路径和相应图片对象关联的软引用的映射关系，在内存不足时，JVM会自动回收这些缓存

图片对象所占有的空间，从而有效的避免OOM问题

虚引用就是在垃圾回收之前，它还能做什么事

WeakhashMap只要key=null就回收

当释放对象内存的时候，会将引用加入到引用队列，当引用队列有数据的时候，一位着引用指向的堆内存的对象被回收了，通过这种方式，JVM允许我们在对象被销毁后，做一些我们想做的事。

83OOM-Overhead limit exceeded(GC回收时间过长，超过0.98的时间用来回收，但是回收不到0.02的堆内存)

OOM-Direct buffer memory(直接内存挂了，元空间使用直接内存，netty,nio（案例：写nio直接分配对外内存，如果不断分配本地内存，堆内存很少使用，jvm就不执行GC,堆外内存不属于GC管辖范围内）)

OOM-unable to create new Native thread(创建线程太多)

OOM-Metaspace(元空间不使用jvm内存，使用的本地内存)

OOM-堆溢出

OOM-栈溢出

92四种垃圾收集器组合方式

Serial-serial old

parnew-cms-serialold

parallel scange-serial old

parallel scavenge-parallel old

100GC如何选择垃圾收集器

单cpu或小内存 -XX:+UseSericalGC

多CPU，最大吞吐量：-XX:+UseParallelGC或者-XX:+UseParallelOldGC

多cpu,最低停顿时间：-XX:+UseCOnCMarkSweepGC，-XX:+ParNewGC

105

top和uptime查看负载情况

vmstat,pidstat查看cpu

free,pidstat查看内存

df查看磁盘

iostat 查看磁盘io

ifstat 查看网络