**Java**

**1.** **new一个对象的过程中发生了什么？**

**2.** **介绍下ClassLoader？什么时候打破双亲委派模式？**

**3.** **为什么等待和通知是在 Object 类而不是 Thread 中声明的？（为什么 wait，notify 和 notifyAll 是在 Object 类中定义的而不是在 Thread 类中定义）**

**4.** **为什么Java中类不支持多重继承？**

**5.** **为什么 String 在 Java 中是不可变的？**

**6.** **编写 Java 程序时, 如何在 Java 中创建死锁并修复它？**

**7. 为什么Java中 wait 方法需要在 synchronized 的方法中调用？**

**8．Object 类中方法及说明？**

**9.接口和抽象类的区别,注意JDK8的接口可以有实现。**

**10.java中传值还是传引用？**

**11．**[**java中String、StringBuffer、StringBuilder的区别**](https://www.cnblogs.com/xudong-bupt/p/3961159.html)

**12.equals方法实现和==**

**13. 面向对象**

**14. Java八种基本数据类型**

**15.Excption与Error包结构，OOM你遇到过哪些情况，SOF你遇到过哪些情况？**

**16.Override（覆盖、重写）和 Overload（重载）的区别？**

**17.java反射的作用与原理**

**18. 如何实现浅克隆和深克隆？**

**19. IntegerX5,inty=5,比较x=y都经过哪些步骤?**

**20. final、finally、finalize的区别？**

**21.Java中的闭包（Closure）**

**22. HashMap、TreeSet、TreeMap、Hashtable、HashSet、LinkedHashMap、ConcurrentHashMap、ArrayList、LinkedList、Vector、CopyOnWriteArrayList、LinkedBlockingQueue的实现和区别之类的。**

**23．List、Map、Set三个接口，存取元素时，各有什么特点。**

**24.如何才能得到一个线程安全的HashMap？**

**25.Java有哪些队列？**

**26.快速失败 (fail-fast) 和安全失败 (fail-safe) 的区别是什么？**

**27.heap 和 stack 有什么区别。**

**28.Comparable 和 Comparator 接口是干什么的？列出它们的区别**

**29.CLH同步队列是怎么实现非公平和公平的？**

**30.COW之CopyOnWriteArrayList、CAS之ConcurrentHashMap（JDK1.8）、读写分离之LinkedBlockingQueue**

**31.HashMap和HashTable有何不同？**

**32.** **ArrayList如何实现扩容？**

**33、Hashcode的作用？**

**34.** **HashMap rehash慢如何解决(渐进式hash、多线程并行rehash)**

**35. Arrays.sort 实现原理和 Collection 实现原理？**

JVM

**1.了解过JVM调优没，基本思路是什么**

**2.淘宝热门商品信息在JVM哪个内存区域**

**3.了解过字节码的编译过程吗？词法解析过程？**

**4.讲一下JVM的内存结构(就是内存五大分区)（也叫内存模型）（内存分配）**

**5.讲一下JVM常用垃圾回收器？**

**6.线上发送频繁full gc如何处理? 如何定位问题?如何解决?说下解决思路和处理方法。**

**7. CPU使用率过高怎么办? 如何定位问题?如何解决?说下解决思路和处理方法。**

**8.class.forName("java.langString")和String.class.getClassLoader().LoadClass("java.lang.String")什么区别?**

**9、Object o = new Object()在内存中占用多少字节？**

**10、对象在内存中的存储布局？对象头具体包括什么？对象怎么定位？**

**11.** **如何判断一个对象是否存活？（或者 GC 对象的判定方法）**

**13. Minor GC与Full GC分别在什么时候发生？**

**14、JVM三大性能调优参数，JVM几个重要的参数?**

**15. Java 中都有哪些引用类型？**

**16.oom3种情况？**

**17.** **对象什么情况下进入老年代？**

**MySQL**

**1.** **讲一讲MySQL的索引结构？**

**2.** **mysql都有什么锁，死锁判定原理和具体场景，死锁怎么解决?**

**3.** **你在做sql优化主要从哪几个方面做？**

**4. 什么是回表，什么是索引覆盖？什么是最左匹配？什么是索引下推？**

**5. 为什么推荐使用自增主键做索引？什么是页分裂，什么是符合并？**

**6. 索引有哪些优化方式呢？**

**7. 事务四大特性？**

**8. 数据库隔离级别，每个级别会引发什么问题，mysql默认是哪个级别？**

**9.** **索引有B+索引和hash索引，各自的区别？**

**10.** **B+索引数据结构，和B树（就是B-树）的区别？**

**11.** **聚集索引和非聚集索引区别？**

**12. 索引的分类（主键索引、唯一索引），哪些情况索引会失效？**

**13. 有哪些锁（乐观锁悲观锁），select时怎么加排它锁？**

**14. 数据库三范式，根据某个场景设计数据表？**

**15. MySQL中有六种日志文件？**

**16. 数据库的读写分离、主从复制？**

**17. 使用explain（执行计划）优化sql和索引？**

**18. mysql并发情况下怎么解决?**

**19.** **分表分库机制？**

**20. Mybatis#和$的区别？**

NIO

**1.I/O、BIO、NIO和AIO的区别？**

**2.NIO的组成？**

**3.** **Netty的特点？**

**4. Netty的线程模型？**

**5.** **Netty的零拷贝实现？**

**6. 什么是同步？什么是异步？阻塞和非阻塞？**

**7. select、poll和epoll的区别？**

**8. TCP 粘包/拆包的原因及解决方法？**

Redis

**1.讲一下redis的数据结构（List和ZSet）?**

**2. redis缓存同步问题?（和数据库保持一致性）**

**3. Redis为什么速度这么快?** **为什么Redis是单线程的？**

**4. Redis扩容机制（渐进式单线程扩容）**

**5.** **redis 过期键的删除策略？从节点过期key是如何删除的呢？**

**6.** **Redis 的同步机制了解么？**

**7.** **Redis缓存穿透，缓存雪崩？**

**8. 如何使用Redis来实现分布式锁？（Lua）**

**9. Redis的并发竞争问题如何解决?（如何解决Redis的并发竞争key问题）（就是加锁、队列）**

**10. Redis持久化的几种方式，优缺点是什么，怎么实现的？**

**11.** **Redis的缓存失效策略？**

**12.** **Hash一致性算法**

**13. Redis的数据淘汰策略？（内存回收策略）**

**14．Redis cluster集群？哈希槽？哨兵模式（sentinel）？管道模式？**

**15.** **Redis和数据库双写一致性问题？**

Spring

**1. Spring 框架中的单例 Beans 是线程安全的么？**

**2. Spring 框架中都用到了哪些设计模式？**

**3.** **aop机制（原理），Spring AOP里面的几个名词**

**4.** **Spring的IoC理解？IOC和DI，依赖注入和控制反转到底什么关系？**

**5.请解释Spring Bean的生命周期？**

**6. FactoryBean和BeanFactory以及普通Bean三者的关系和区别？**

**7.请举例解释@Autowired注解？**

SpringBoot

**1. 讲一下Springboot的启动流程吧?**

**2.** **SpringBoot配置文件的加载顺序？**

Spring MVC

**1. SpringMVC的流程？**

**2.** **SpringMvc的控制器是不是单例模式,如果是,有什么问题,怎么解决？**

TCP

**1.** **TCP/IP、HTTP协议？**

**2. Http请求的过程与原理？(一次 HTTP 请求的全过程，包括域名解析、定位主机等？)**

**3.** **三次握手，四次挥手？为什么TCP连接需要三次握手，两次不可以吗？**

**4. Tcp和udp区别？**

**5. DNS在网络层用哪个协议，为什么？CDN实现原理？**

**6.** **介绍HTTPS协议，详述SSL建立连接过程**

**7.** **TCP的三次握手与四次挥手过程，各个状态名称与含义，TIMEWAIT的作用?** **为什么A在TIME-WAIT状态必须等待2MSL的时间？为什么连接的时候是三次握手，关闭的时候却是四次握手？**

**8. 电脑上访问一个网页，整个过程是怎么样的：DNS、HTTP、TCP、OSPF、IP、ARP？**

**9.** **对称加密 和 非对称加密，滑动窗口协议**

Zk

**1.** **消息广播的实现原理？写请求？ZAB协议？**

**2.** **集群角色？集群组成？Leader选举？高可用原理？**

**3.** **zookeeper数据结构？zk的通知机制？（watcher）**

**4.分布式锁？**

**5. 测试和生产共用一套zookeeper，怎么保证消费不冲突?**

**6. Zookeeper和Eureka区别？**

多线程

**1.** **wait和sleep区别？**

**2.** **线程同步的方法。多线程同步和互斥有几种实现方法？**

**3.** **synchronized与lock的区别，使用场景。看过synchronized的源码没？**

**4.** **volatile？**

**5.** **java锁有哪些种类，以及区别？什么是锁消除和锁粗化？**

**6.** **线程池的参数，线程池的执行原理以及各种线程池的应用场景？**

**7.** **ReentrantLock 其实现原理？AQS？如何实现可重入性的？**

**8.** **除了 ReetrantLock，你还接触过 JUC 中的哪些并发工具？**

**9.** **请谈谈 ThreadLocal 是怎么解决并发安全的？使用 ThreadLocal 需要注意些什么？**

**10.** **在 java 中守护线程和本地线程区别？**

**11.** **为什么使用 Executor 框架？（就是线程池优点）**

**12. 什么是CAS？**

**13. 线程的五个状态（五种状态，创建、就绪、运行、阻塞和死亡）?**

**14.** **线程实现的方式有几种（四种）？**

**15.** **锁的等级：方法锁、对象锁、类锁?**

**16. 如何检测死锁？怎么预防死锁？**

高并发，低耗时的情况：建议少线程，只要满足并发即可；例如并发100，线程池可能设置为10就可以  
低并发，高耗时的情况：建议多线程，保证有空闲线程，接受新的任务；例如并发10，线程池可能就要设置为20；  
高并发高耗时：1要分析任务类型，2增加排队，3、加大线程数

Dubbo

**1. Dubbo中zookeeper做注册中心，如果注册中心集群都挂掉，发布者和订阅者之间还能通信么？**

**2.** **dubbo服务负载均衡策略？默认是随机，可以再xml里更改配置。**

**3.** **dubbo服务集群配置（集群容错模式）**

**4.** **画一画服务注册与发现的流程图**

**5.** **dubbo 基本原理**

MQ

**1.使用了消息队列会有什么缺点？**

**2.** **如何保证消息队列是高可用的？**

**3.** **如何保证消息不被重复消费？**

**4.** **如何保证消费的可靠性传输？（不丢消息）**

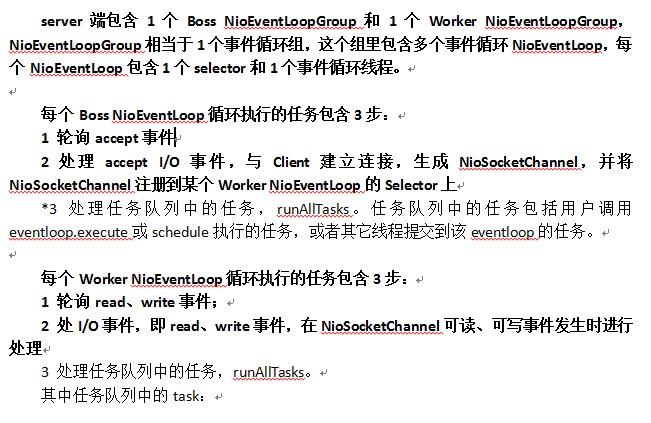
**5.下游消费系统如果宕机了，导致几百万条消息在消息中间件里积压**

**6.** **RocketMQ很大的一个特点是对分布式事务的支持，你说说他在分布式事务支持这块机制的底层原理?**

**设计模式？**

**Lambda？**

**数据结构？算法？**



与select相比，epoll分清了频繁调用和不频繁调用的操作。例如，epoll\_ctrl是不太频繁调用的，而epoll\_wait是非常频繁调用的。

epoll是通过内核与用户空间mmap同一块内存实现的。**mmap将用户空间的一块地址和内核空间的一块地址同时映射到相同的一块物理内存地址**（不管是用户空间还是内核空间都是虚拟地址，最终要通过地址映射映射到物理地址），**使得这块物理内存对内核和对用户均可见，减少用户态和内核态之间的数据交换**

**红黑树将存储epoll所监听的套接字**。

**epoll使用“事件”的就绪通知方式，通过epoll\_ctl注册fd，一旦该fd就绪，内核就会采用类似callback的回调机制来激活该fd，epoll\_wait便可以收到通知**)。**当相应的事件发生后**，就会调用这个回调函数，该回调函数在内核中被称为：ep\_poll\_callback,**这个回调函数其实就所把这个事件添加到rdllist这个双向链表中**。一旦有事件发生，epoll就会将该事件添加到双向链表中。那么当我们调用epoll\_wait时，**epoll\_wait只需要检查rdlist双向链表中是否有存在注册的事件**

**HashSet：**HashSet是哈希表实现的。HashSet中的数据是无序的。可以放入null，但只能放入一个null。

**TreeSet：**TreeSet是二差树实现的。Treeset中的数据是自动排好序的。不允许放入null值。

**Hashtable：**同样是基于哈希表实现的，同样每个元素是一个key-value对，其内部也是通过单链表解决冲突问题，容量不足（超过了阀值）时，同样会自动增长。线程安全。不允许存入null key和value。

**ThreadPoolExecutor**

**高并发、任务执行时间短**的业务，线程池线程数可以设置为**CPU核数+1（当其中一个线程遇到内存缺页异常，+1那条线程马上补上）**，减少线程上下文的切换(建议少线程，只要满足并发即可；例如并发100，线程池可能设置为10就可以)

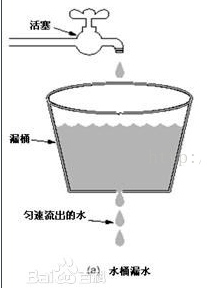
**b. 并发不高、任务执行时间长的业务怎样使用线程池?**

对于耗时长的任务，要分是cpu任务，还是io等类型的任务。如果是**cpu类型**的任务，**线程数不宜太多，保证有空闲线程就行**，例如并发10，线程池可能就要设置为20；但是如果是io类型的任务(**io不费cpu**)，**线程多一些更好，可以更充分利用cpu**。

长连接多用于操作频繁，点对点的通讯，而且连接数不能太多情况，例如：数据库的连接用长连接， 如果用短连接频繁的通信会造成socket错误，而且频繁的socket 创建也是对资源的浪费。

通过idea的maven依赖分析将不需要的依赖exclude掉，打开maven的pom.xml，在某个<dependency>中通过右键菜单：maven->show dependency 打开分析的图形化页面，如下所示：

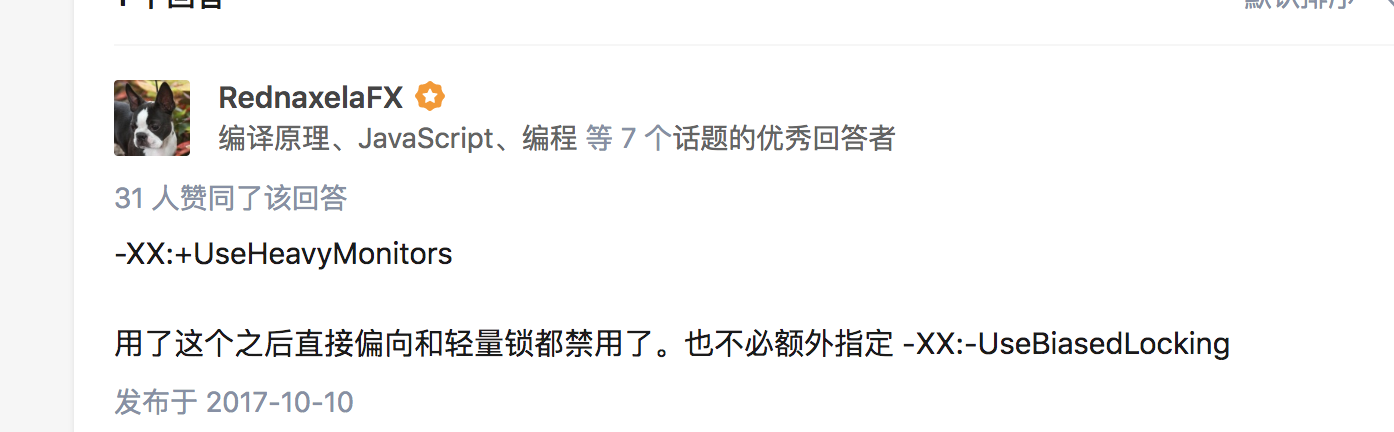
请求先进入到漏桶里，漏桶以一定的速度出水，当水请求过大会直接溢出，可以看出漏桶算法能强行限制数据的传输速率。把请求比作是水，水来了都先放进桶里，并以限定的速度出水，当水来得过猛而出水不够快时就会导致水直接溢出，即拒绝服务。



漏桶算法与令牌桶算法的区别在于，漏桶算法能够强行限制数据的传输速率，令牌桶算法能够在限制数据的平均传输速率的同时还允许某种程度的突发传输。

需要注意的是，在某些情况下，漏桶算法不能够有效地使用网络资源，因为漏桶的漏出速率是固定的，所以即使网络中没有发生拥塞，漏桶算法也不能使某一个单独的数据流达到端口速率。因此，漏桶算法对于存在突发特性的流量来说缺乏效率。而令牌桶算法则能够满足这些具有突发特性的流量。通常，漏桶算法与令牌桶算法结合起来为网络流量提供更高效的控制。

两者主要区别在于“漏桶算法”能够强行限制数据的传输速率，而“令牌桶算法”在能够限制数据的平均传输速率外，还允许某种程度的突发传输。在“令牌桶算法”中，只要令牌桶中存在令牌，那么就允许突发地传输数据直到达到用户配置的门限，所以它适合于具有突发特性的流量。

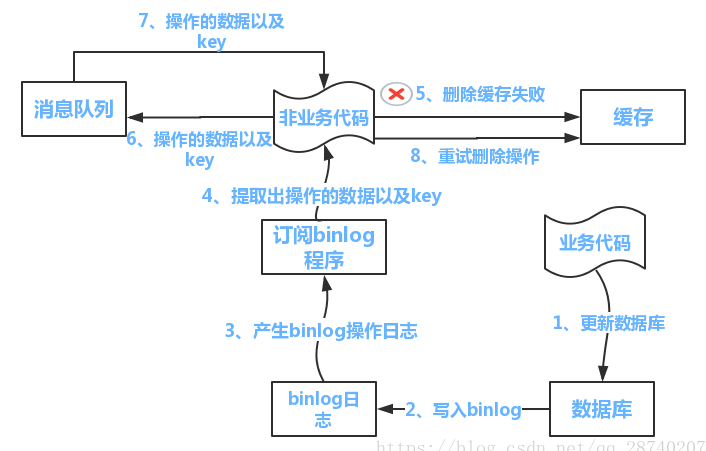




因为在某些情况下父类加载器需要委托子类加载器去加载class文件。受到加载范围的限制，父类加载器无法加载到需要的文件，以Driver接口为例，由于Driver接口定义在jdk当中的，而其实现由各个数据库的服务商来提供，比如mysql的就写了MySQL Connector，那么问题就来了，DriverManager（也由jdk提供）要加载各个实现了Driver接口的实现类，然后进行管理，但是DriverManager由启动类加载器加载，只能记载JAVA\_HOME的lib下文件，而其实现是由服务商提供的，由系统类加载器加载，这个时候就需要启动类加载器来委托子类来加载Driver实现，从而破坏了双亲委派，这里仅仅是举了破坏双亲委派的其中一个情况。

**工厂模式的好处？**

1. 工厂模式是为了**解耦**：可以将对象的创建和使用分离，如果不分离，不但违反了设计模式的**开闭原则**，需要需要使用另一个子类的话，需要修改源代码 ,把对象的创建和使用的过程分开。就是Class A 想调用 Class B ，那么A只是调用B的方法，而至于B的实例化，就交给工厂类。
2. **工厂模式可以降低代码重复**。
3. 因为工**厂管理了对象的创建逻辑，使用者并不需要知道具体的创建过程**，只管使用即可，减少了使用者因为创建逻辑导致的错误。
4. 方法是**一片代码的封装，当方法功能升级的时候，直接去改方法中的代码**，所有调用此方法名的都会产生改变，而不是找到找到每一处的那个不使用方法的一行行功能实现，一个个改，，，这样一算得多麻烦。



保证redis和缓存一致性

（1）更新数据库数据

（2）数据库会将操作信息写入binlog日志当中

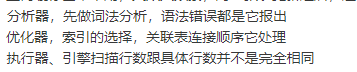
（3）订阅程序提取出所需要的数据以及key

（4）另起一段非业务代码，获得该信息

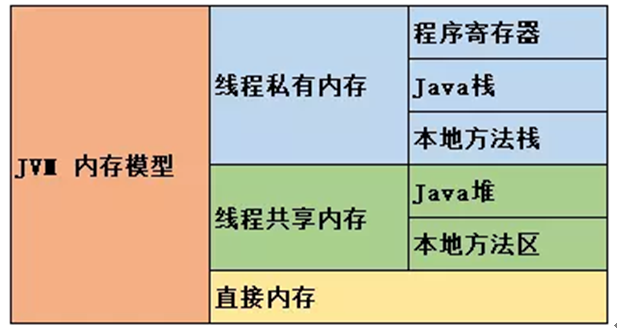
（5）尝试删除缓存操作，发现删除失败

（6）将这些信息发送至消息队列

（7）重新从消息队列中获得该数据，重试操作。



这些器 优化器都在server层，innodb引擎不做



本质上布隆过滤器是一种数据结构，比较巧妙的概率型数据结构（probabilistic data structure），特点是高效地插入和查询，可以用来告诉你 “某样东西一定不存在或者可能存在”。

相比于传统的 List、Set、Map 等数据结构，它更高效、占用空间更少，但是缺点是其返回的结果是概率性的，而不是确切的。

布隆过滤器是一个 bit 向量或者说 bit 数组

果我们要映射一个值到布隆过滤器中，我们需要使用多个不同的哈希函数生成多个哈希值，并对每个生成的哈希值指向的 bit 位置 1，例如针对值 “baidu” 和三个不同的哈希函数分别生成了哈希值 1、4、7，

Netty的服务端使用了两个EventLoopGroup，而第一个EventLoopGroup通常只有一个EventLoop，通常叫做bossGroup，负责客户端的连接请求，然后打开Channel，交给后面的EventLoopGroup中的一个EventLoop来负责这个Channel上的所有读写事件。。

那么对于读写事件的监听是交给worker

**频繁fullgc**：有大量内存垃圾不断在老年代产生；也许是你的大对象（缓存）过多；也有可能是你的参数设置不好，minor GC清理不掉内存，导致每次minor GC都会触发Full GC；还有可能是你的老年代大小参数设置错误，老年代过小等等原因

内存泄漏：在Java中，内存泄漏就是存在一些被分配的对象，这些对象有下面两个特点：

这些对象是可达的，即在有向图中，存在通路可以与其相连；

这些对象是无用的，即程序以后不会再使用这些对象。

如果对象满足这两个条件，这些对象就可以判定为Java中的内存泄漏，这些对象不会被GC所回收，然而它却占用内存。



默认的，新生代 ( Young ) 与老年代 ( Old ) 的比例的值为 1:2 ( 该值可以通过参数 –XX:NewRatio 来指定 )，即：新生代 ( Young ) = 1/3 的堆空间大小。老年代 ( Old ) = 2/3 的堆空间大小。其中，新生代 ( Young ) 被细分为 Eden 和 两个 Survivor 区域，这两个 Survivor 区域分别被命名为 from 和 to，以示区分。  
默认的，Eden : from : to = 8 : 1 : 1 ( 可以通过参数 –XX:SurvivorRatio 来设定 )，即： Eden = 8/10 的新生代空间大小，from = to = 1/10 的新生代空间大小。  
JVM 每次只会使用 Eden 和其中的一块 Survivor 区域来为对象服务，所以无论什么时候，总是有一块 Survivor 区域是空闲着的。  
因此，新生代实际可用的内存空间为 9/10 ( 即90% )的新生代空间。

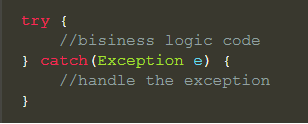
将派生于Error或者RuntimeException的异常称为unchecked异常，所有其他的异常成为checked异常。

如果出现了RuntimeException，就一定是程序员自身的问题。比如说，数组下标越界和访问空指针异常等等，只要你稍加留心这些异常都是在编码阶段可以避免的异常。

spring事务只会对Error与RuntimeException及其子类这些UNChecked异常，做出回滚。 一般的Exception这些Checked异常不会发生回滚（如果一般Exception想回滚要做出配置）；

**将异常捕获,并且在catch块中不对事务做显式提交(或其他应该做的操作如关闭资源等)=生吞掉异常**

spring的事务边界是在调用业务方法之前开始的，业务方法执行完毕之后来执行commit or rollback(Spring默认取决于是否抛出runtime异常).   
如果抛出runtime exception 并在你的业务方法中没有catch到的话，事务会回滚。   
一般不需要在业务方法中catch异常，如果非要catch，在做完你想做的工作后（比如关闭文件等）一定要抛出runtime exception，否则spring会将你的操作commit,这样就会产生脏数据.所以你的catch代码是画蛇添足。   
**由此可以推知，在spring中如果某个业务方法被一个**



**整个包裹起来，则这个业务方法也就等于脱离了spring事务的管理，因为没有任何异常会从业务方法中抛出！全被捕获并吞掉，导致spring异常抛出触发事务回滚策略失效。   
不过，如果在catch代码块中采用页面硬编码的方式使用spring api对事务做显式的回滚，这样写也未尝不可。**

**spring 事务处理中，同一个类中:A方法（无事务）调B方法（有事务）,事务不生效问题?**

**在一个Service内部，事务方法之间的嵌套调用，普通方法和事务方法之间的嵌套调用，都不会开启新的事务.是因为spring采用动态代理机制来实现事务控制，而动态代理最终都是要调用原始对象的，而原始对象在去调用方法时，是不会再触发代理了！**

Sharding和mycat

就是重新实现JDBC的API，通过重新实现DataSource、PrepareStatement等操作数据库的接口，让应用层在基本（注意：这里用了基本）不改变业务代码的情况下透明地实现分库分表的能力。

中间件给上层应用提供熟悉的JDBC API，内部通过sql解析、sql重写、sql路由等一系列的准备工作获取真正可执行的sql，然后底层再按照传统的方法（比如数据库连接池）获取物理连接来执行sql，最后把数据结果合并处理成ResultSet返回给应用层。

因此一般乐观锁只用在高并发、多读少写的场景。

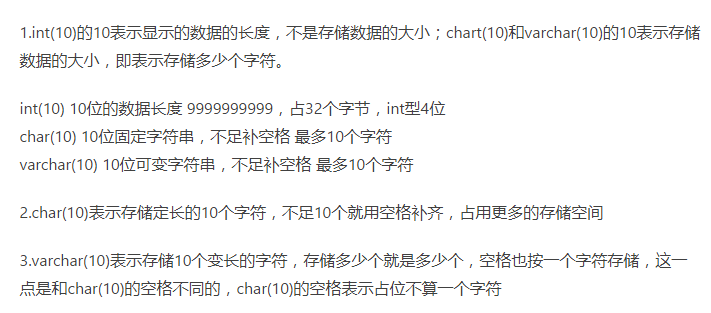
**什么是内存泄漏和内存溢出？**

（溢出是指创建太多对象导致内存空间不足，泄漏是无用对象没有回收）

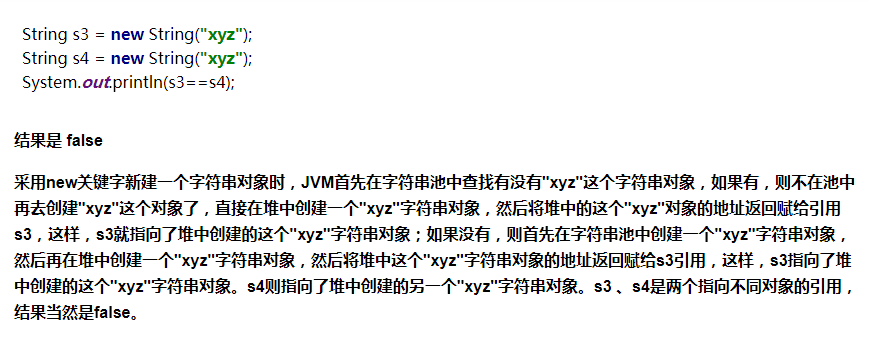
**java类加载之加载类的三种途径？**

1. **由 new 关键字创建一个类的实例**（静态加载）
2. **调用 Class.forName() 方法**  
   通过反射加载类型，并创建对象实例
3. **调用某个 ClassLoader 实例的 loadClass() 方法**  
   通过该 ClassLoader 实例的 loadClass() 方法载入。应用程序可以通过继承 ClassLoader 实现自己的类装载器。

**varchar（10） 和int（11）这俩代表啥？**



运行时常量池相对于Class文件常量池的另外一个重要特征是具备动态性，Java语言并不要求常量一定只有编译期才能产生，也就是并非预置入Class文件中常量池的内容才能进入方法区运行时常量池，运行时常量池，运行期间也可能将新的常量放入池中，这种特性被开发人员利用比较多的就是String类的intern()方法。



**检测死锁？**

通过jvisualvm查看产生死锁线程

jstack命令生成的thread dump信息包含了JVM中所有存活的线程，为了分析指定线程，必须找出对应线程的调用栈，应该如何找？

在top命令中，已经获取到了占用cpu资源较高的线程pid，将该pid转成16进制的值，在thread dump中每个线程都有一个nid，找到对应的nid即可；隔段时间再执行一次stack命令获取thread dump，区分两份dump是否有差别，在nid=0x246c的线程调用栈中，发现该线程一直在执行JstackCase类第33行的calculate方法，得到这个信息，就可以检查对应的代码是否有问题。

**Dump文件方法？**

Jvm配置自动dump，linux jmap dump文件

在jvm启动的参数中，新增-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/home/admin/logs/java.hprof jvm参数。这样在**发生jvm 内存溢出时，就会直接dump出java.hprof 文件**了。

直接导出jvm内存信息。

jmap -dump:format=b,file=/home/admin/logs/heap.hprof javapid

**mysql事务怎么实现？**