题目来源：https://juejin.im/post/5d5217246fb9a06ae61aabf5

1.StringBuffer，StringBuilder区别是啥？

StringBuffer是线程安全的，而StringBuilder是非线程安全的。

2.什么是线程安全？

线程安全是编程中的术语，指某个函数、函数库在并发环境中被调用时，能够正确地处理多个线程之间的共享变量，使程序功能正确完成。即在多线程场景下，不发生有序性、原子性以及可见性问题。

3.如何保证线程安全？

Java中主要通过加锁来实现线程安全。通常使用synchronized和Lock

4.什么是锁？死锁？

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

死锁的发生必须具备以下四个必要条件：互斥条件、请求和保持条件、不剥夺条件、环路等待条件

死锁的解决办法就是破坏以上四种必备条件中的一个或者多个。

5.synchronized的实现原理是什么？

深入理解多线程（一）——Synchronized的实现原理 深入理解多线程（四）—— Moniter的实现原理 再有人问你synchronized是什么，就把这篇文章发给他

6.有了synchronized，还要volatile干什么？

volatile通常被比喻成”轻量级的synchronized“，volatile可以保证可见性和有序性，实现原理是通过内存屏障实现的。

volatile有一个重要的作用，是synchronized不具备的，那就是禁止指令重排序。这一特点在双重校验锁实现单例的时候有用到，虽然使用了synchronized关键字，但是如果不用volatile修饰单例对象，就会存在问题。

深入理解Java中的volatile关键字 再有人问你synchronized是什么，就把这篇文章发给他。

7.synchronized的锁优化是怎么回事？（锁粗化？锁消除？自旋锁？偏向锁？轻量级锁？）

深入理解多线程（五）—— Java虚拟机的锁优化技术

8.知道JMM吗？（原子性？可见性？有序性？）

Java内存模型（Java Memory Model ,JMM）是一种符合内存模型规范的，屏蔽了各种硬件和操作系统的访问差异的，保证了Java程序在各种平台下对内存的访问都能保证效果一致的机制及规范。

再有人问你Java内存模型是什么，就把这篇文章发给他。

9.Java并发包了解吗？

java.util.concurrent包(J.U.C)中包含的是java并发编程中有用的一些工具类，包括几个部分：

1、locks部分：包含在java.util.concurrent.locks包中，提供显式锁(互斥锁和速写锁)相关功能；

2、atomic部分：包含在java.util.concurrent.atomic包中，提供原子变量类相关的功能，是构建非阻塞算法的基础；

3、executor部分：散落在java.util.concurrent包中，提供线程池相关的功能；

4、collections部分：散落在java.util.concurrent包中，提供并发容器相关功能；

5、tools部分：散落在java.util.concurrent包中，提供同步工具类，如信号量、闭锁、栅栏等功能；

10.那什么是fail-fast？什么是fail-safe？

我们通常说的Java中的fail-fast机制，默认指的是Java集合的一种错误检测机制。当多个线程对部分集合进行结构上的改变的操作时，有可能会产生fail-fast机制，这个时候就会抛出ConcurrentModificationException。

ConcurrentModificationException，当方法检测到对象的并发修改，但不允许这种修改时就抛出该异常。

为了避免触发fail-fast机制，导致异常，我们可以使用Java中提供的一些采用了fail-safe机制的集合类。

这样的集合容器在遍历时不是直接在集合内容上访问的，而是先复制原有集合内容，在拷贝的集合上进行遍历。

java.util.concurrent包下的容器都是fail-safe的，可以在多线程下并发使用，并发修改。同时也可以在foreach中进行add/remove 。

一不小心就踩坑的fail-fast是个什么鬼？

11.什么是CopyOnWrite？

Copy-On-Write简称COW，是一种用于程序设计中的优化策略。其基本思路是，从一开始大家都在共享同一个内容，当某个人想要修改这个内容的时候，才会真正把内容Copy出去形成一个新的内容然后再改，这是一种延时懒惰策略。

CopyOnWrite容器即写时复制的容器。通俗的理解是当我们往一个容器添加元素的时候，不直接往当前容器添加，而是先将当前容器进行Copy，复制出一个新的容器，然后新的容器里添加元素，添加完元素之后，再将原容器的引用指向新的容器。

一不小心就踩坑的fail-fast是个什么鬼？

12.那AQS呢？那CAS呢？

AQS(AbstractQueuedSynchronizer)，即队列同步器。它是构建锁或者其他同步组件的基础框架（如ReentrantLock、ReentrantReadWriteLock、Semaphore等），JUC并发包的作者（Doug Lea）期望它能够成为实现大部分同步需求的基础。它是JUC并发包中的核心基础组件。

CAS是项乐观锁技术，当多个线程尝试使用CAS同时更新同一个变量时，只有其中一个线程能更新变量的值，而其它线程都失败，失败的线程并不会被挂起，而是被告知这次竞争中失败，并可以再次尝试。

CAS 操作包含三个操作数 —— 内存位置（V）、预期原值（A）和新值(B)。如果内存位置的值与预期原值相匹配，那么处理器会自动将该位置值更新为新值。否则，处理器不做任何操作。无论哪种情况，它都会在 CAS 指令之前返回该位置的值。（在 CAS 的一些特殊情况下将仅返回 CAS 是否成功，而不提取当前值。）CAS 有效地说明了“我认为位置 V 应该包含值 A；如果包含该值，则将 B 放到这个位置；否则，不要更改该位置，只告诉我这个位置现在的值即可。”这其实和乐观锁的冲突检查+数据更新的原理是一样的。

乐观锁的一种实现方式——CAS

13.CAS都知道，那乐观锁一定知道了？

乐观锁（ Optimistic Locking ） 相对悲观锁而言，乐观锁假设认为数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让返回用户错误的信息，让用户决定如何去做。

相对于悲观锁，在对数据库进行处理的时候，乐观锁并不会使用数据库提供的锁机制。一般的实现乐观锁的方式就是记录数据版本。

实现数据版本有两种方式，第一种是使用版本号，第二种是使用时间戳。

深入理解乐观锁与悲观锁

14.乐观锁悲观锁区别是什么？

同上

15.数据库如何实现悲观锁和乐观锁？

深入理解乐观锁与悲观锁

16.数据库锁有了解么？行级锁？表级锁？共享锁？排他锁？gap锁？next-key lock？

MySQL中的行级锁,表级锁,页级锁

MySQL中的共享锁与排他锁

17.数据库锁和隔离级别有什么关系？

很多DBMS定义了多个不同的“事务隔离等级”来控制锁的程度和并发能力。

ANSI/ISO SQL定义的标准隔离级别有四种，从高到底依次为：可序列化(Serializable)、可重复读(Repeatable reads)、提交读(Read committed)、未提交读(Read uncommitted)。

深入分析事务的隔离级别

18.数据库锁和索引有什么关系？

在MySQL中，行级锁并不是直接锁记录，而是锁索引。索引分为主键索引和非主键索引两种，如果一条sql语句操作了主键索引，MySQL就会锁定这条主键索引；如果一条语句操作了非主键索引，MySQL会先锁定该非主键索引，再锁定相关的主键索引。

19.什么是聚簇索引？非聚簇索引？最左前缀是什么？B+树索引？联合索引？回表？

主键索引的叶子节点存的是整行数据。在InnoDB中，主键索引也被称为聚簇索引（clustered index）

非主键索引的叶子节点的内容是主键的值，在InnoDB中，非主键索引也被称为非聚簇索引（secondary index）

当我们创建一个联合索引的时候，如(key1,key2,key3)，相当于创建了（key1）、(key1,key2)和(key1,key2,key3)三个索引，这就是最左匹配原则。

在 InnoDB 里，索引B+ Tree的叶子节点存储了整行数据的是主键索引。而索引B+ Tree的叶子节点存储了主键的值的是非主键索引。因为主键索引树的叶子节点直接就是我们要查询的整行数据了。而非主键索引的叶子节点是主键的值，查到主键的值以后，还需要再通过主键的值再进行一次查询，这个过程叫做回表。

[我以为我对Mysql索引很了解，直到我遇到了阿里的面试官]

20.分布式锁有了解吗？

目前比较常用的有以下几种方案：

基于数据库实现分布式锁 基于缓存（redis，memcached，tair）实现分布式锁 基于Zookeeper实现分布式锁

[分布式锁的几种实现方式~]

21.Redis怎么实现分布式锁？

多个进程执行以下Redis命令：

SETNX lock.foo

如果 SETNX 返回1，说明该进程获得锁，SETNX将键 lock.foo 的值设置为锁的超时时间（当前时间 + 锁的有效时间）。 如果 SETNX 返回0，说明其他进程已经获得了锁，进程不能进入临界区。进程可以在一个循环中不断地尝试 SETNX 操作，以获得锁。

22.为什么要用Redis？

分布式缓存，提升性能

23.Redis和memcache区别是什么？

1、存储方式：Memcache把数据全部存在内存之中，断电后会挂掉，数据不能超 过内存大小。Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启时可以再次加载进行使用。（RDB快照和AOF日志两 种持久化方式）。

2、Redis支持数据的备份，及master-slave模式的数据备份。

3、数据支持类型：Redis在数据支持上要比Memcache多得多。

4、使用底层模型不同：新版本的Redis直接自己构建了VM机制，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

24.Zookeeper怎么实现分布式锁？

基于zookeeper临时有序节点可以实现的分布式锁。

大致思想即为：每个客户端对某个方法加锁时，在zookeeper上的与该方法对应的指定节点的目录下，生成一个唯一的瞬时有序节点。 判断是否获取锁的方式很简单，只需要判断有序节点中序号最小的一个。 当释放锁的时候，只需将这个瞬时节点删除即可。同时，其可以避免服务宕机导致的锁无法释放，而产生的死锁问题。

分布式锁的几种实现方式~

25.什么是Zookeeper？

Zookeeper是一个开放源码的分布式服务协调组件，是Google Chubby的开源实现。是一个高性能的分布式数据一致性解决方案。他将那些复杂的、容易出错的分布式一致性服务封装起来，构成一个高效可靠的原语集，并提供一系列简单易用的接口给用户使用。

Zookeeper介绍（二）——Zookeeper概述

26.什么是CAP？

CAP理论：一个分布式系统最多只能同时满足一致性（Consistency）、可用性（Availability）和分区容错性（Partition tolerance）这三项中的两项。

分布式系统的CAP理论

27.什么是BASE？和CAP什么区别？

BASE理论是对CAP理论的延伸，核心思想是即使无法做到强一致性（Strong Consistency，CAP的一致性就是强一致性），但应用可以采用适合的方式达到最终一致性（Eventual Consitency）。

BASE是指基本可用（Basically Available）、软状态（ Soft State）、最终一致性（ Eventual Consistency）。

分布式系统的BASE理论

28.CAP怎么推导？如何取舍？

对于涉及到钱财这样不能有一丝让步的场景，C必须保证。网络发生故障宁可停止服务，这是保证CP，舍弃A。比如前几年支付宝光缆被挖断的事件，在网络出现故障的时候，支付宝就在可用性和数据一致性之间选择了数据一致性，用户感受到的是支付宝系统长时间宕机，但是其实背后是无数的工程师在恢复数据，保证数数据的一致性。

对于其他场景，比较普遍的做法是选择可用性和分区容错性，舍弃强一致性，退而求其次使用最终一致性来保证数据的安全。

分布式系统的CAP理论

29.分布式系统怎么保证数据一致性？

分布式事务

30.啥是分布式事务？分布式事务方案？

分布式事务是指会涉及到操作多个数据库的事务。其实就是将对同一库事务的概念扩大到了对多个库的事务。目的是为了保证分布式系统中的数据一致性。分布式事务处理的关键是必须有一种方法可以知道事务在任何地方所做的所有动作，提交或回滚事务的决定必须产生统一的结果（全部提交或全部回滚）

关于分布式事务、两阶段提交协议、三阶提交协议

分布式事务解决方案——柔性事务与服务模式

那么，最后了，来手写一个线程安全的单例吧？

单例模式的七种写法

为什么我墙裂建议大家使用枚举来实现单例

不用synchronized和lock能实现线程安全的单例吗？

借助CAS（AtomicReference）实现单例模式：

```
public class Singleton {
    private static final AtomicReference<Singleton> INSTANCE = new
AtomicReference<Singleton>();

    private Singleton() {}

    public static Singleton getInstance() {
        for (;;) {
            Singleton singleton = INSTANCE.get();
            if (null != singleton) {
                return singleton;
            }

            singleton = new Singleton();
            if (INSTANCE.compareAndSet(null, singleton)) {
                return singleton;
            }
        }
    }
}
```

复制代码

用CAS的好处在于不需要使用传统的锁机制来保证线程安全,CAS是一种基于忙等待的算法,依赖底层硬件的实现,相对于锁它没有线程切换和阻塞的额外消耗,可以支持较大的并行度。

CAS的一个重要缺点在于如果忙等待一直执行不成功(一直在死循环中),会对CPU造成较大的执行开销。

不使用synchronized和lock，如何实现一个线程安全的单例？

不使用synchronized和lock，如何实现一个线程安全的单例？（二）

这你都能答上？那好吧，你给我解释下什么是Paxos算法吧？

Paxos一种基于消息传递且具有高度容错特性的一致性算法。Paxos算法号称是最难理解的算法！！！