java 面试题汇总

==========算法需要注意

==========

1. **进制转换：405.** java 负数取余是负数。
2. **消除游戏：390.**
3. **打乱数组：384.**
4. **O(1)时间插入删除随机获取元素：380.** remove 时的顺序要小心。
5. **有序矩阵第 k 小**：378。n 路归并
6. **组合总和 IV**：377。顺序可以重复计算的情况下，要换种 dp 模式。
7. **摆动序列：376。**up/down 表示以 0 到 i 内，而不是以 i 结尾。
8. **水壶问题：365。**ax+by=c 有解 => c%gcd(a,b)=0。gcd 的算法要记住。
9. **计算各个位数不同的数字个数：357。**
10. **摆动排序 2：324。**分成两部分，反转再穿插。
11. **最大单词长度乘积：318。**快速判断两个单词是否有公共字母：按位与。
12. **去除重复字母：316。**单调栈。单调栈适合需要维持一个递增序列，且 pop 元素对最终结果无影响的情况。
13. **最大间距：164。**找到一个数的相邻数使用 treeset，lower / higher 或者 ceiling / floor。
14. **excel 表列名称：168**。减 1 然后对 26 运算。
15. **颠倒二进制位：190。**
16. **数字范围内按位与：201。**即求公共前缀。
17. **存在重复元素 iii：220。**treeset + 滑动窗口。
18. **求众数 ii：229**。先判断是否相同，再判断是否个数为 0。
19. **去除重复字母：316**。单调栈。

20. **买卖股票 iii：123。**动态规划，buy1，sell1，buy2，sell2。

21. **三角形最小路径和：120。**动态规划，从下往上加，要注意。

==========java 语法==========

1. jdk 和 jre 的区别

jre 是 java runtime environment，可以运行 java 程序，jdk 实际包含 jre，除了 jre 之外还有编译器 javac，以及调试等工具，因此只要安装 jdk 即可。

jdk7 和 jdk8 有什么区别：支持 lambda 表达式(函数式编程)，支持 stream api，hashMap 性能优化等。

lambda 表达式主要用于简化匿名内部类(实现只有一个函数的接口)的代码等，如堆的新建。

stream api 可以方便简洁的进行遍历操作，例如

```
myArray.stream().filter(xx).sorted().forEach(xx)
```

2. == 和 equals 的区别是什么

==在基本类型时是值的比较，但是对于引用类型，是比较两个引用是否一致。

equals 是 Object 类(一切类的父类)的方法，可以用于引用类型的值比较。

3. final 在 java 中有什么作用？

用 final 修饰的类，不能被继承

用 final 修饰的方法，不能被重写

用 final 修饰的变量，称为常量，值在初始化后不能再被修改

Integer 是引用传递吗：

Integer 是引用传递，但是内部声明是 final int，所以每次对 Integer 修改都会生成新的对象，不能将 Integer 用于作共享变量，String 也一样。

4. 说说 static 关键字

静态方法：不依赖于任何对象，通过类名即可访问。

静态变量：在方法区分配内存，只有一个副本，被类的所有对象所共享。

静态代码块：只在初始化的时候执行一次，优先于构造函数执行

静态内部类：独立于外部类存在，只是借用一下壳子，为了组织方便。（内部类：依赖于外部类存在，创建需要依赖外部类(也有 private、public 等权限的区别)，可以自由访问外部类的成员）。

5. String、StringBuffer、StringBuilder 的区别

String 是不可变对象(内部是一个 char 数组，并且用 final 修饰)，每次操作都会生成新的对 象，因此频繁变更时应该避免使用 String。

StringBuffer 是可变对象，是线程安全的，但效率略低，适合多线程。

StringBuilder 是可变对象，不是线程安全的，但是效率更高，适合单线程。

6. java 抽象类是什么

抽象类是为了把相似但是还未确定的东西进行抽取，定义抽象类的目的，就是为了在子类中对抽象方法进行不同的实现。

特点：

1. 有抽象方法的一定是抽象类，但是抽象类可以不包含抽象方法。
2. 抽象类不能被实例化，因为没有意义。
3. 如果子类不是抽象类，则必须实现全部的抽象方法。

6. java 接口是什么

接口的是一种规范，是最高级别的抽象。主要作用是为不同类之间的交互提供标准，同时实现类之间的解耦。假设一个接口下有几个不同的实现类，在另一个类想要使用这些类的时候，我们无需知道每个类的具体实现代码，只需要根据接口的定义进行使用即可，这样可以大

大提高编程效率以及代码的简洁度。

特点：

1. 接口不是类，不能实例化
2. 接口不能有构造方法，所有的方法必须为抽象方法，必须是 public abstract
3. 接口不能有成员变量，除非 public static final
4. 一个类可以实现多个接口，但是只能继承一个类

接口和抽象类是不同层次的抽象，最根本的属性也不同，共同致力于代码的高效简洁。

6：重写和重载的区别

重写是子类重写父类的方法，方法名，参数，返回值必须一样，只能重写 public 和 protected 修饰的方法，final 修饰的方法不能重写。

重载是指一个类中方法名相同，但是参数不同，返回值也不同的方法，比如构造方法。

6. 讲讲封装继承多态

三者都是面向对象的特征。

封装是指一个类将内部变量和实现细节隐藏起来，只对外保留相应的接口(比如 set、get)。

继承是指子类可以继承父类非 private 修饰的变量和方法，这样可以将公共的部分提取出来，便于面向对象的组织。

多态由重写和重载实现，可以在不同情况下灵活的表现为对应的状态，比如不同的构造函数。

7. java 中的 IO 流都有哪些，BIO、NIO、AIO 有什么区别

1. 按功能分为输入流和输出流
2. 按类型分为字节流和字符流
3. 按是否同步和是否阻塞分为 BIO(同步阻塞)、NIO(同步非阻塞)、AIO(异步非阻塞)

同步和异步是针对客户端获取结果的方式而言的，同步需要线程主动的查看是否有 IO 结果，异步通过回调函数的方式主动通知给线程。

阻塞和非阻塞是针对等待结果期间线程行为而言的，阻塞是线程不去做别的，一直等待结果返回，非阻塞是指线程去做别的任务。

8. java 中 error 和 exception 的区别

error 和 exception 都是 throwable 类的子类。error 是 jvm 抛出的错误，用户通过代码无法处理这类错误，比如 OutOfMemoryError；exception 是用户代码层面抛出的异常，是可以通过代码来解决的。

9. java 中 checkedException 和 uncheckedException 的区别

checkedException 代表程序无法控制的外界异常，比如 IOException，SQLException 等，所有的 checkedException 都必须被 throws 或者 try catch，否则在编译期 java 就会提示错误。

uncheckedException 代表程序本身存在问题，比如 NPE，IndexOutOfBoundException 等，uncheckedException 无需显式的抛出异常。

值得一提的是，checkedException 是 java 自身的一种设计，目前普遍被认为是不必要的，因为很多情况下需要写大量的 try catch，影响程序的简洁性。

子类抛出的异常类型能不能比父类抛出的异常类型更宽泛：

子类可以抛出任意 uncheckedException，但是对于 checkedException，子类不能比父类更宽泛，例如父类声明抛出 IOException，子类不能声明抛出 Exception。这样设计是为了维护面向对象程序的规范性，将子类的异常控制在一定的范围内。

10. java 中的 throw 和 throws 的区别

throw 是代码主动抛出一个具体的异常，throws 是方法声明自己可能抛出哪些异常。如果是 checkedException 的话都需要上一层调用函数进行 try catch 处理或者继续抛出。

11. java try catch finally 的执行顺序

正常的执行顺序是 try catch finally，如果在 try 中有 return 语句，finally 中的代码依然

会执行，因为在 jvm 看来 return 语句也只是一个普通的返回，但是如果 try 中有 System.exit(1)的话 finally 中的代码不会执行，因为此时进程已经退出。

11. 描述以下 java 中的反射机制

java 反射是指程序在**运行时能够获取自身的信息**的一种机制，给定类的名字，就可以通过反射机制获取类的所有信息、构造对象、调用方法。常见的使用方式是 Class c = Class.forName(xxx)。原理是所有的类编译后都有一个.class 文件，反射其实是在运行时通过类的加载并执行一系列操作来完成的，详见 class.forName 的原理。

优点是动态加载，非常灵活，可以在运行时动态的加载不同的类。

缺点是性能较差。

11. class.forName 和 classLoader 的区别是什么

在 java 中两者都可以进行类的加载，classLoader 作为类加载器将类的字节码.class 文件加载到 jvm 内存中，并不执行后续的初始化操作，只有在 newInstance 时才会执行初始化操作，而 class.forName 在此基础上还会执行初始化操作(执行 static 语句等)，class.forName 也是调用了 classLoader 实现了第一步。

jdbc 中 class.forName(xxx.Driver)背后会执行静态代码块用于注册一个 Driver。

11. 什么是 java 序列化？什么情况下需要序列化？

序列化是指将对象转变为字节流的过程，反序列化是指将字节流转换成对象的过程。

当我们需要通过网络传输对象或者将对象持久化到文件中时，需要进行序列化操作。

注意，被 transient 关键字修饰的变量不能序列化，一般而言，如果一些变量只是临时变量，我们不希望序列化它(占用空间)，可以使用 transient 关键字。但是构造方法中的变量不能加 transient 关键字，否则就无法反序列化。

同时静态变量不能序列化，因为静态变量是类的属性，而序列化针对的是对象。

java 序列化要实现 Serializable 接口，serialVersionUID 相当于版本号，目的是**保证接收方与发送方所操作的对象版本是一致的**。在进行反序列化时，接收方会将字节流中的

serialVersionUID 与本地类中的 serialVersionUID 进行比较，如果不同则报错。

11. java 深拷贝和浅拷贝

深拷贝和浅拷贝都是创建一个新对象，但是对象内的成员变量处理方式不同。

浅拷贝是将成员变量的引用地址进行拷贝，指向同一个内存地址，一个对象改变后另一个对象也跟着改变。

深拷贝会创建一个新对象并将值进行拷贝，一个对象的改变不影响另一个对象。

11：Object 类中有哪些方法

toString()：返回该对象的描述。

wait()：让当前线程进入阻塞状态并且释放锁，直到其它线程调用 notify 方法唤醒。

notify()，notifyAll()：对等待的线程进行唤醒，注意，**wait 方法会释放锁，这两个方法会唤醒线程进行抢锁操作，抢到锁才可以继续运行，否则继续抢锁。**

equals()：判断是否相等。

hashcode()：返回哈希值，用于 equals 判断。

getClass()：获取对象对应的类（和 class.forName 不同的是后者根据类名获取，会进行类加载的工作，而前者根据对象获取，不会进行类加载的工作）。

clone()：浅拷贝。浅拷贝是指新建一个对象，但是对象里面使用引用传递。

finalize()：gc 时会调用该方法。

注意：每个类都有一个 class 对象用于描述自身，放在方法区。

重写 equals 为什么要重写 hashcode：

equals 和 hashcode 都是判断两个对象是否相等，前者是准确判断，后者是近似判断(性能更高)，equals 的两个对象 hashcode 必定相等，hashcode 相等 equals 不一定相等。如果不重写 hashcode 可能不满足第一个条件，也就不符合 java 的约定。

11. 内存溢出和内存泄漏的区别

内存溢出是指内存超出了容量，比如 stackoverflow。

内存泄露是指不需要的内存没有被正确的释放，导致该内存一直被占用。

11：说说 java 泛型

泛型的好处是：1. 提高了代码的复用性 2：避免了类型的强制转换等。

11. java 中的闭锁和栅栏有什么区别

都是线程进行同步的方式，但是使用场景不同：

闭锁是一个或多个线程等待其他线程执行完毕(调用 countDown)，再运行程序。

栅栏时线程之间相互等待，直到全部达到了阈值时再一起运行后面的程序，使用一个同步辅助类（CycleBarrier）。

https://segmentfault.com/a/1190000022941013

说说 countDownLatch 的实现原理：

countDownLatch 是闭锁的实现方式，内部使用 AQS 实现，初始化的时候将 count 赋值给 state，调用 await 方法的线程会被放入阻塞队列，每执行一次 countDown 操作 state 就会减 1，state 为 0 时会进行线程的唤醒。（栅栏的实现也类似，计数器达到相应的值即放开栅栏）。

11. 说说 java 的动态代理

静态代理是实现一个接口并进行实例化，但是如果有很多类的话会很繁琐。

动态代理是 jdk 提供的一个能力可以在运行时动态的实例化一个接口，背后的机制与反射相似，jvm 会创建类并动态加载。

12. 什么情况下会发生 StackOverFlow 和 OutOfMemory

StackOverFlow 是栈溢出，比如无限递归的时候就会出现 StackOverFlow。

OutOfMemory 是堆溢出，在堆中分配的对象超出堆的大小时就会 OutOfMemory。

==========java 多线程==========

12. java 创建线程的方式

1. 继承 Thread 类并实现 run 方法
2. 实现 runnable 接口(也是 run 方法)，实例化后作为构造参数创建 Thread 对象

13. synchronized 关键字的作用

synchronized 是 java 的关键字，利用锁的机制来实现同步。

java 中的锁分为对象锁和类锁，对象锁是针对某一对象加锁，有两种方式，一种是对方法加锁，例如声明为 public synchronized void xxx () {}，另一种是对代码块加锁，例如 synchronized(this) {}。对该对象而言，被 synchronized 修饰的部分同一时间只允许一个线程访问，其他部分仍然可以并发。

类锁是对类加锁，一种方式是对静态方法加锁，例如声明 public synchronized static void xxx(){}，另一种方式是对代码块加锁，例如 synchronized(myClass.class) {}。所有该类实例化后的对象，在同一时间只能有一个线程访问被 synchronized 修饰的部分，其他部分仍然可以并发。

synchronized 不能继承，需要在子类中显式的使用 synchronized 关键字；接口方法不能使用 synchronized，构造方法不能使用 synchronized，但是构造方法中可以使用 synchronized 修饰代码块进行同步。

参考：https://juejin.cn/post/6844903482114195463

注意，如果要在多个方法之间控制并发，可以使用一个全局变量 Object 对象，对其进行加解锁即可，详见 leetcode 多线程题目。

Synchronized 原理

Synchronized 是基于进入和退出监视器(moniter)对象实现的，通过编译后会在同步代码块前后插入 monitorenter 和 monitorexit 两个指令，在执行 monitorenter 指令时，会首先尝试获取 moniter 对象的控制权，成功则将计数器+1，否则阻塞等待，执行 monitorexit 时会将计数器-1。在 java 的对象头中存在指向 moniter 对象的指针，通过该指针访问 moniter 对象。

Synchronized 慢的原因是线程阻塞会进行用户态和内核态之前的切换，因此比 volatile

慢。

14. volatile 关键字的作用

总结：volatile 可以保证可见性和有序性，涉及复合操作的变量不宜使用 volatile，synchronized 可以同时满足原子性、可见行和有序性，volatile 性能优于 synchronized。

目前的内存模型是主存+cache，线程从主存中读取变量到 cache 中，操作之后再写回 cache，cache 再写回主存，基于这个背景，在并发编程中要满足三个特性：**原子性、可见性、有序性**。

原子性指一条指令要么全执行，要么全不执行，a=1 这样的指令就是原子的，a++这样的指令就是非原子的，因为它包含从主存中读，运算、写回主存三个步骤。

可见行指的是一个线程对数据的更改，对另一个线程应该是可见的，因为数据从缓存写回主存需要一定的时间，因此可能造成一个变量的写操作对另一个变量不可见。

有序性指的是指令之间的有序性，cpu 为了加速运算，会存在指令重排的情况，在多线程情况下可能出错。

volatile 修饰的变量可以保证可见性和有序性，当一个线程对变量进行修改后，会立即写回主存，保证其他线程能读到最新的值；同时使用 volatile 修饰的变量可以避免指令重排。

注意，volatile 修饰的变量无法保证原子性，因此如果变量有复合操作(比如 x++)，不能使用 volatile 修饰。volatile 背后使用内存屏障来保证可见性和有序性，synchronized 可以同时保证三个特性，但是 synchronized 采用锁的机制会性能会比 volatile 差。

volatile 的原理：

可见性实现：变量修改后强制将变量刷回主存，保证其他线程读取该数时是正确的。

有序性实现：添加内存屏障来避免指令重排序，内存屏障之前的指令执行完后面的才能执行。

15. synchronized 和 Lock 的区别

1. synchronized 是 java 的关键字，不需要手动开启和释放锁；而 Lock 是一个接口，用户可以有更灵活的实现，同时需要手动加解锁。

2. synchronized 只支持非公平锁，ReentrantLock 可以同时支持非公平锁和公平锁，两者都是可重入的。
3. synchronized 等待加锁的线程不能中止等待，而 ReentrantLock 的等待可中断，等待时间过长时可以中断等待去做别的任务。
4. ReentrantLock 可以设置等锁的超时时间，这样不容易出现死锁，而 synchronized 必须一直等待。

15. Lock 的底层原理：

ReentrantLock 底层核心组件是 AQS(抽象队列同步器)，是一个类，核心并发操作都由它完成，核心变量是 state(int，记录状态，初始为 0)、当前获得锁的线程、阻塞队列。当一个线程加锁时，将 state+1(CAS)，并且更新当前获得锁的线程，如果此期间有线程尝试加锁，则将其放入阻塞队列。解锁时将 state-1(CAS)，当前获得锁的线程置为 null。

这也就解释了为什么可重入(有当前线程的标识)，为什么可以同时实现公平锁和非公平锁(阻塞队列)。

参考：https://zhuanlan.zhihu.com/p/86072774

Condition 的主要 api 是 await，signal 和 signalAll，与 object 类似。

15. 什么是可重入锁，公平锁和非公平锁

可重入锁是指一个线程获取锁之后可以再次进入该段程序(例如递归调用)，java 内置的 synchronized 和 ReentrantLock 都是可重入锁。

公平锁指的是严格按顺序排序获取锁，非公平锁是指如果释放锁时正好有新的线程来获取锁，则直接交给它，否则进入队列排队。这样的好处是避免了线程的上下文切换，减少开销。

ReentrantLock 可重入利用了 AQS 的 state，如果是同一个线程访问则+1，离开-1，为 0 的时候释放。

16. java 中的 atomic 类型是什么

java 中的 atomic 类型是为了解决普通类型中无法保证原子性问题而引入的，在多线程的并发编程中，atomic 类型可以保证复合操作的原子性，例如自增操作，同时使用 atomic 类型比使用 synchronized 的开销要小。

atomic 类型底层的原理是 volatile+CAS 原理，是线程安全的。

17. 讲讲 CAS 原理

CAS(compareAndSet)，比较并赋值，当**内存中的值**与**期望值**相同时，则 set 为**更新值，**否则根据内存值更新期望值，进行自旋操作。

因为非原子性的问题，线程工作内存中的值可能和主存中不一致，此时如果直接写回会导致主存数据错误，因此采用 CAS 的方式保证原子性。

CAS 的原理是 unsafe 类和自旋锁，unsafe 类(可调用操作系统 api)会调用更底层的 compareAndSet 的 api，自旋锁会连续进行加锁尝试。

CAS 的优点是和加锁相比提高了并发。

缺点是：1. 自旋操作会造成开销较大

2. 只能保证一个变量的原子性
3. 会产生 ABA 问题(原来为 A，改为 B 后又改为 A，解决方式是加版本号)。

17. ThreadLocal 是什么

ThreadLocal 是线程局部变量，是独属于一个线程的，不能被其他线程访问或者修改，这就为多线程情况下变量的隔离提供了可能。我们将目标变量封装进 ThreadLocal，则每个 thread 都对自己所拥有的变量进行操作，实现了线程之间的隔离。

synchronized 是实现目标变量在多线程之间的同步，而 ThreadLocal 是实现目标变量在多线程之间的隔离。

ThreadLocal 背后的原理是使用 ThreadLocalMap(最基本的 api 是 set 和 get)，key 是 ThreadLocal，value 是对应的值，其中 key 是弱引用，value 是强引用，这就导致了内存泄漏问题：ThreadLocal 被回收了，而 value 依然存在，直到该线程的生命周期结束 value 才被回收，如果是线程池的话就会造成内存泄漏问题，解决方法是使用完后主动的调用 remove 方法进行释放。

18. sleep 方法和 wait 方法的区别是什么

1. sleep 是 Thread 类的方法，可以随时使用，wait 是 Object 类的方法，只能在同步代码块中使用。

2. 都使当前线程进入阻塞状态，但是 sleep 不释放锁，wait 释放锁
3. sleep 需要等待阻塞时间到达后自动唤醒，wait 方法可以通过 notify 和 notifyAll 两种方式主动唤醒

19. 线程的 run()和 start()有什么区别

线程的 run 方法中是主线程调用具体的运行代码，并不启动线程，而 start 真正用于启动一个线程至就绪状态，当 cpu 调度到当前线程时，再运行 run 中的代码。

19. 线程的生命周期是怎么样的

线程的生命周期包括创建、就绪、运行、阻塞、结束。

19. 讲讲 java 中的偏向锁、轻量级锁和重量级锁

syncronized 进入和退出 monitor 对象的方式属于重量级锁，开销较大，因此引入了偏向锁和轻量级锁进行优化。

偏向锁：减少无实际竞争情况下，轻量级锁的性能损耗。一个线程获取锁之后不会释放，下次可以直接进入同步代码块，直到有竞争时会膨胀成轻量级锁。偏向锁只获取一次锁，而轻量级锁每次都需要一个 CAS 操作加锁和解锁。

轻量级锁：减少无实际竞争情况下，重量级锁开销大的问题(内核态与用户态的翻转)。方法是每次通过 CAS 更新对象头中的记录值，如果失败则使用自旋锁重试，自旋锁尝试失败则膨胀为重量级锁。

重量级锁：基于进入和退出 monitor 对象实现。

20. 自旋锁、自适应自旋锁、锁消除、锁粗化

自旋锁：是指线程获取锁失败后原地等待，定时再次请求，而不进入阻塞态。对于每次加锁后使用时间较短的场景而言，自旋锁避免了进入阻塞态之后用户态到内核态的翻转，大大提高性能。但是对于加锁之后用时很久的场景而言，这种方式会大大消耗性能。

自适应自旋锁：请求加锁的时间不再固定，根据情况动态决定。

锁消除：如果 jvm 检测到不可能出现竞争，则不再加锁。

锁粗化：如果检测到连续的加锁和解锁操作，则将锁优化成更大范围的锁来降低开销。

20. java 线程池是什么，重要的参数有哪些

线程池是用来管理线程的池子，通过避免线程的频繁创建和销毁来提高性能。

重要的参数包括：

1. 核心线程的数量
2. 最大线程的数量(包括核心线程和非核心线程)
3. 非核心线程存活时间
4. 存放任务的队列
5. 拒绝策略
6. 创建线程的工厂

21：阻塞队列有哪几种

1. 有界队列：队列容量有限
2. 无界队列：队列容量无限
3. 同步队列：容量是 0，只进行任务的传递
4. 优先队列：支持按优先级排序
5. 延迟队列：可以设置任务延迟多久后执行

20. 常用的有哪几种

newFixedThreadPool：全是核心线程，阻塞队列采用无界队列，适合 cpu 密集型任务。

newCachedThreadPool：全是非核心线程，线程数目不存在限制，空闲线程会被回收，队列采用同步队列，适合 IO 密集型任务。

newSingleThreadExecutor：单线程的线程池，串行执行任务，阻塞队列采用有界队

列，适合串行任务。

newScheduledThreadPool：定时调度的线程池，阻塞队列采用延迟队列，适合定时任务。

参考：https://juejin.cn/post/6844903889678893063#heading-43

21：线程池阻塞队列满了之后的拒绝策略有哪些：

1. 抛异常
2. 直接丢弃
3. 丢弃阻塞队列中最老的任务
4. 不进入线程池，由当前调用线程进行运行

21. 线程池中 submit()和 execute()方法有什么区别

1. execute 没有返回值；submit 有返回值 Future，通过 get 方法可以获得异常，因此 submit 方法方便异常处理。
2. execute 只能提交 runnable 的任务，submit 可以提交 runnable 和 callable 的任务。

22. 线程池怎么做异常处理

线程池中的异常是不会被直接抛出来的，有两种处理方式：

1. 使用 try catch
2. 使用 submit 提交任务，之后调用 future.get 获取异常。

22. 线程池的状态都有哪些

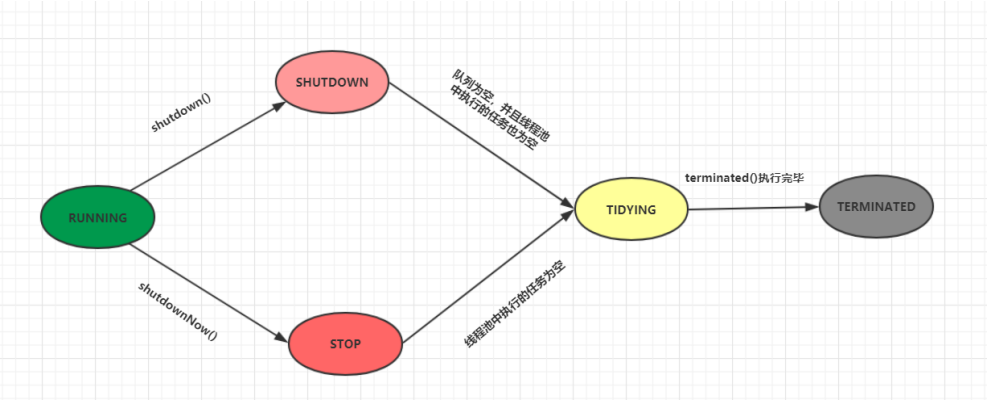
running：运行中的线程池

shutdown：不会接受新任务，但是会处理在队列中的任务

stop：停止处理一切任务

tidying：所有的任务已终止

terminated：线程池彻底结束



22：线程池中线程的数量由什么确定

计算密集型的任务中线程数量=cpu 核心数，IO 密集型任务中线程数量=2*cpu 核心数。

22：多线程编程

全局变量 count=0，1 个主线程打印 start 后，多个子线程按顺序对 count+1，并打印出 count 值，count==n 时，主线程打印 end 并退出。

参考 CountToN。

参考 https://zhuanlan.zhihu.com/p/370130458。

==========设计模式==========

26. java 中常见的设计模式都有哪些

单例模式：一个类只有一个对象，并且作为类的静态变量常驻，主要用来在只需要一个对象的场景下，减少对象频繁的创建和销毁来提高性能。一般将构造方法声明为 private 来防止实例化，同时通过静态变量的形式保证唯一，实现的时候有两种方式：懒汉式(第一次获取实例时创建)和饿汉式(主动创建)。

- **饿汉**
  预先创建，可能存在资源浪费

- **懒汉**
  使用时再创建，并发情况下可能会创建多次，因此需要加类锁
- **双重校验锁**
  先判断是否存在再加类锁，提高了效率，但是可能因为指令优化出现失效
- **静态内部类**
  在静态内部类中创建静态对象，因为静态内部类只有在使用时才会被加载，这样既解决了懒汉模式中的资源浪费问题，又解决了饿汉模式中的并发问题。

如何破坏单例：

如果构造方法被声明为 private，那么无法正常的实例化，可以通过反射(设置构造函数为可访问)或者序列化反序列化的方式破坏单例。

工厂模式：工厂模式提供了一种创建对象的最佳实践。一般流程是定义一个接口，不同的类对接口进行实现，然后工厂类**根据传入参数的不同选择不同的类进行实例化并返回**，相当于提供了一个统一的接口用于创建对象。在 flink connector 中一般都有一个 Factory 类，会根据情况选择创建 source、sink 或者 dim。

抽象工厂模式：相当于在工厂模式上又加了一层，定义了一个抽象类，各个不同的工厂类继承该抽象类，使用时会根据参数的不同创建不同的工厂，然后就回到了工厂模式。相当于制造工厂的工厂。

建造者模式(builder 模式)：builder 模式是在类中提**供了一个静态内部类 Builder 用来负责创建对象，这样可以将对象的创建过程单独分离**，在参数很多的情况下可以大大提高程序的简洁性。当一个类的成员变量很多且部分必须部分不必须时，需要定义非常多的构造方法，非常繁琐，如果对每个变量提供 set 方法来进行初始化，会造成对象随时被更改的问题，而 builder 可以很好的解决这些问题。

观察者模式：当对象间存在一对多的关系时，就使用观察者模式，即将观察该类的对象都加入该类的一个成员变量 List 中，当该类发生变化时，即通知所有的对象。

==========jvm==========

27. jvm 都由哪些部分组成，各个部分的作用是什么

jvm 的组成部分包括类加载器、运行时数据区、执行引擎、本地库接口。

java 代码首先被编译成字节码(.class 文件)，类加载器将字节码加载到运行时数据区并进行一系列加载操作，随后执行引擎将字节码翻译成指令交给 cpu 去处理，这个过程中有些功能需要用到本地库接口。

28. jvm 运行时数据区的组成

程序计数器：保存当前线程所执行的字节码的行号，每个线程都有自己的程序计数器。

虚拟机栈：虚拟机栈是用来描述 java **方法**执行的内存模型，存放着局部变量，方法入口出口等信息，每个线程都有自己的虚拟机栈。

本地方法栈：与虚拟机栈类似，是为 java 以外的方法准备的。

堆：最大的一块内存，所有线程共享，存放对象实例，由垃圾回收器进行回收。

方法区：方法区用来存放编译后的代码、常量、静态变量等，所有线程共享。

29. java 中堆和栈的区别

栈是描述 java 方法执行的内存模型，其中存储的是局部变量以及方法的出入口等，当方法执行完毕时会立刻进行回收。

堆是主要用来存储 java 中的对象，由垃圾回收器进行回收。

30. 类加载的过程是怎么样的

首先一个完整的类生命周期是编译——加载——使用——回收。

加载过程包括加载、验证、准备、解析、初始化。

加载：将类的字节码加载到内存中。

验证：一些格式等是否符合规范。

准备：为 static 变量开辟内存空间并用默认值初始化。

解析：把符号引用替换为直接引用(符号替换为实际的地址)。

初始化：对 static 修饰的变量或语句进行初始化。

31. 双亲委派模型是怎么样的

双亲委派模型是类加载机制的一部分。

双亲委派模型指的是除了顶层的启动类加载器外，其他的加载器都应当有自己的父加载器，当加载一个类时，应该首先交给父加载器，如果父加载器无法加载，则子加载器再进行加载。

对于任何一个类，都需要加载器和这个类本身来确定唯一性，这样设计的好处是保证一些重要的基类不受破坏，例如我们自定义了一个 java.lang.Object，但是因为他不是启动类加载器加载的，所以无法进行覆盖，这也就保证了程序的安全性。

目前有系统提供了三种类加载器，由上到下分别是：**启动类加载器、扩展类加载器、应用程序类加载器。**

32. 讲讲 java 垃圾回收

java 垃圾回收主要是指堆区和方法区的回收，要弄清楚三个问题，什么样的对象可以被回收、什么时候进行回收，如何回收。

1. 什么样的对象可以被回收

判断对象是否可以被回收的有两种算法：

a. **引用计数算法：**判断对象的被引用数量，为 0 则被回收。

引用计数法的缺陷：1. 计数器占用空间，且变化非常频繁 2. 循环引用的情况无解

b. **可达性分析算法：**将程序中所有的引用关系建立一张图，通过判断对象在图中是否可达来进行判断

2. 什么时候进行回收

java 堆区一般分为新生代、老年代、永久代三部分。

新生代用来存放生命周期短的对象，又分为 eden 区和两个 survive 区，eden 区用来新建对象，其中一个 survive 区存放之前的存活对象，另一个 survive 区存放 gc 后的存活对象，每次 minor gc 之后两个 survive 区会互换角色，这样的复制算法可以减少内存碎片，当 survive 区不够用时就将对象放到老年代。

新生代进行的 gc 叫 minor gc，频率较高；全局都进行的 gc 叫 full gc，老年代满了或者永久代满了时才会触发，消耗的时间较长。

永久代存放静态变量和常量等，如果永久代满了，会触发 full gc(针对全局的垃圾回收)。

新生代和老年代比例默认 1：2，新生代 eden 和两个 survive 默认 8:1:1。

3. 如何回收

新生代采用的是复制算法(分为两块相互导)，如前所述；老年代采用的是标记-整理算法，即先对存活的对象进行标记，随后将标记过的对象移动至一端，随后清理其他。

32. 什么对象可以放入老年代，进入老年代阈值是多少

大对象和在新生代中存活较久的对象可以进入老年代，进入老年代的阈值可以由参数进行设定。

32. jvm 垃圾回收器都有哪些

串行垃圾回收器：单线程进行垃圾回收，应用线程全部停止，适合非常简单的程序。

并行垃圾回收器：多线程进行垃圾回收，应用线程全部停止，适合对阻塞时间要求短的程序。

CMS 垃圾回收器：多线程回收，优点在于停顿低，在垃圾回收的同时可以保证应用程序的运行效率，因为其采用标记-清除算法并进行了细粒度的拆分，缺点是内存碎片多。

G1 垃圾回收器：将堆内存分成一个个块区域，打破了代间的物理隔离。与 CMS 相比：

1. 采用标记-整理算法，减少了内存碎片。
2. 停顿时间可预测
3. 更充分的利用多核 cpu 来减少停顿时间
4. 包含 young gc 和 mixed gc(回收一部分老年代来减少停顿)，没有 full gc，而 CMS 只能作用于老年代。

32：哪些可以当做 GC root

gc root 是指在标记判断对象是否可达时，最原始的入口，如果从 gc root 不能到达对象，则该对象会被回收。

虚拟机栈中局部变量引用的对象，方法区中类的静态变量、常量引用的对象等。

32. 常见的 GC 调优策略

最终目标都是减少 full gc 的次数。包括：

1. 调节新生代的大小，尽量减少 full gc 的次数。
2. 合理设置进入老年代的年龄，合理设置直接进入老年代的对象大小阈值。
3. 合理设置堆的大小。

jvm 怎么调优：

首先观察 jvm 的运行参数和运行状态，比如通过 jps、jstack 命令或者图形化界面了解进程、线程、内存、cpu、gc 等信息，然后根据情况进行调优，调优的一个主要的方向是 gc 调优。

33. java 中的几种引用是什么

java 中的引用由强到弱依次是：强引用 -> 软引用 -> 弱引用 -> 虚引用。

强引用的对象永远不会被 gc，软引用的对象内存不足时会 gc，弱引用的对象随时会 gc，虚引用的对象好像没有引用一样。

34. java 对象头包括什么

java 对象包括对象头和实例数据，对象头中记录了对象相关的信息，包括：

1. mark word：记录了对象锁信息(偏向/轻量/重量)和 gc 信息(代数)。
2. 指向类信息的指针：类信息在方法区中。
3. 数组长度：如果对象是数组的话有该项。

35. 讲讲 happens-before

happens-bofore 是 jmm 的一种设计，为了保证可见性。如果一个操作优先于另一个操作，则第一个操作先执行，且结果对第二个操作可见。在不影响结果的前提下，可以进行指令重排序。

==========**java 数据结构**==========

33. 说说 java 中的集合类

Map 接口和 Collection 接口是所有集合的父接口，其中 Collection 接口的子接口包括 List 接口和 Set 接口。

Map 接口主要的实现类有：HashMap、Hashtable、TreeMap、ConcurrentHashMap。

List 接口主要的实现类有：ArrayList，LinkedList，Vector。

34. HashMap、Hashtable、TreeMap 的实现有什么不同

HashMap 底层数据结构是哈希表，采用链表法解决冲突，当链表过长时会转换成红黑树。HashTable 与 HashMap 类似，不同之处在于：

1. HashMap 是线程不安全的，HashTable 加了 synchronized 关键字，是线程安全的。
2. HashMap 的 key 和 value 都可以是 null，HashTable 的 key 和 value 都不可以是 null。
3. 某些 api 不一样。

而 TreeMap 底层是红黑树，可以通过实现比较器的方法来进行排序。而基于 hash 表的方式元素都是无序的，LinkedHashMap 可以按照插入的顺序保证有序。

35. 聊一下 ConcurrentHashMap

hashMap 线程不安全，hashTable 使用 synchronized 效率较低，多线程情况下推荐使用 ConcurrentHashMap。ConcurrentHashMap 主要采用分段锁技术，将数据先分段，不同段之间的操作互不影响，而不是像 hashTable 一样全局一把锁。

36：HashMap Put 过程

1. 初始化数组，会自动设定为比传入参数大的最近的 2 的幂次方。

2. 对 key 进行哈希计算，判断对应下标是否有数据，没有则插入。
3. 如果有的话就进行链表/红黑树的遍历，有相同 key 则覆盖，没有则新建。
4. 判断是否超过阈值，超过则进行扩容操作。

36：HashMap 扩容时，为什么扩两倍

在 put 计算位置时，公式是(len - 1) & hash(key)。只有 len 是 2 的幂次方时，len-1 全为 1，可以充分散列，减少碰撞。

为什么扩容的负载因子是 0.75:

当数据超过 0.75 * len 的时候就会触发扩容操作，0.75 是在时间效率和空间效率上做了平衡，在牺牲部分空间效率的基础上，避免了大量的哈希冲突，提高时间效率。

36：JDK 1.8 HashMap 有哪些变化

1. 数据结构：jdk1.7 中 HashMap 采用数组+链表的数据结构，而 jdk1.8 中 HashMap 采用数组链表+红黑树的数据结构，当链表长度超过 8 时会转变为红黑树。
2. 扩容：jdk1.7 扩容采用 rehash，对每个节点都要重新计算新 hash 值，jdk1.8 中直接利用了数学规律，每个节点只可能是 hash 值和 hash 值+原数组长度，通过计算首位的 0/1 即可。
3. jdk1.7 采用头插法，jdk1.8 采用尾插法。

36：HashMap 为什么线程不安全

jdk1.7 中线程不安全体现在数据丢失和扩容时链表成环导致死循环问题。

jdk1.8 中线程不安全体现在数据丢失(两个 key 的 hash 值相同，完成碰撞检测后准备插入，此时时间片结束，线程 b 完成插入，此时 a 再插入，那么 b 的值就会被覆盖)(采用 table[i]=new Node()方式)。

37. ArrayList、LinkedList、Vector 的区别

ArrayList 是由可变数组(数组，但是能动态扩容)实现的，Vector 也一样，但是 ArrayList 本身是线程不安全的，而 Vector 是线程安全的(加了 synchronized)。LinkedList 底层是双向链表，相比之下插入删除更快，但是查询更慢。

ArrayList 的扩容机制采用 Array.copyOf，会新建一个更大的数组并将元素进行复制。

38. HashSet 和 TreeSet 的区别

HashSet 背后使用的是 HashMap，key 就是数据的值，value 为 null，元素是无序的。

TreeSet 背后使用的是 TreeMap，元素可以进行排序(自定义 + 自然排序)。

==========spring==========

1. 讲讲 spring ioc 和 aop

spring 作为一个轻量级的 java 开发框架，可以简化程序，敏捷开发，其核心思想是 ioc 和 aop。spring 主要负责对象创建、对象管理、依赖管理，通过配置文件的方式进行书写，启动时会根据配置文件新建所有对象。

ioc：控制反转，将对象(bean)的创建和维护交给 spring 容器进行，而不在程序中，主要的作用是简化程序，提高复用等。依赖注入与 ioc 是一个概念，以前调用方需要创建被调用者实例，现在被调用方实例由 spring 容器创建，并且注入调用方，称作依赖注入。

aop：面向切面编程，是面向对象编程的补充。面向对象是自上而下的，而面向切面是在一个切面上的，比如很多对象都有的日志打印相关代码，就可以通过 aop 的方式进行复用。背后的原理是动态代理。