java

解释内存中的栈(stack)、堆(heap)和方法区(method area)的用法。

答：通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用JVM中的栈空间；而通过new关键字和构造器创建的对象则放在堆空间，堆是垃圾收集器管理的主要区域，由于现在的垃圾收集器都采用分代收集算法，所以堆空间还可以细分为新生代和老生代，再具体一点可以分为Eden、Survivor（又可分为From Survivor和To Survivor）、Tenured；方法区和堆都是各个线程共享的内存区域，用于存储已经被JVM加载的类信息、常量、静态变量、JIT编译器编译后的代码等数据；程序中的字面量（literal）如直接书写的100、"hello"和常量都是放在常量池中，常量池是方法区的一部分，。栈空间操作起来最快但是栈很小，通常大量的对象都是放在堆空间，栈和堆的大小都可以通过JVM的启动参数来进行调整，栈空间用光了会引发StackOverflowError，而堆和常量池空间不足则会引发OutOfMemoryError。

String和StringBuilder、StringBuffer的区别？

答：Java平台提供了两种类型的字符串：String和StringBuffer/StringBuilder，它们可以储存和操作字符串。其中String是只读字符串，也就意味着String引用的字符串内容是不能被改变的。而StringBuffer/StringBuilder类表示的字符串对象可以直接进行修改。StringBuilder是Java 5中引入的，它和StringBuffer的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被synchronized修饰（非同步），因此它的效率也比StringBuffer要高。

重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分？

答：方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。重载发生在一个类中，同名的方法如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；重写发生在子类与父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常（里氏代换原则）。重载对返回类型没有特殊的要求

描述一下JVM加载class文件的原理机制？

双亲委派模型

抽象类（abstract class）和接口（interface）有什么异同？

答：抽象类和接口都不能够实例化，但可以定义抽象类和接口类型的引用。一个类如果继承了某个抽象类或者实现了某个接口都需要对其中的抽象方法全部进行实现，否则该类仍然需要被声明为抽象类。接口比抽象类更加抽象，因为抽象类中可以定义构造器，可以有抽象方法和具体方法，而接口中不能定义构造器而且其中的方法全部都是抽象方法。抽象类中的成员可以是private、默认、protected、public的，而接口中的成员全都是public的。抽象类中可以定义成员变量，而接口中定义的成员变量实际上都是常量。有抽象方法的类必须被声明为抽象类，而抽象类未必要有抽象方法。

Java中会存在内存泄漏吗？请简单描述。

答：理论上Java因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是Java被广泛使用于服务器端编程的一个重要原因）；然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被GC回收，因此也会导致内存泄露的发生。例如Hibernate的Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象，如果不及时关闭（close）或清空（flush）一级缓存就可能导致内存泄露。下面例子中的代码也会导致内存泄露。

Java 中的final关键字有哪些用法？

(1)修饰类：表示该类不能被继承；

(2)修饰方法：表示方法不能被重写；

(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）

什么时候用断言（assert）？

答：断言在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，断言用于保证程序最基本、关键的正确性。断言检查通常在开发和测试时开启。为了保证程序的执行效率，在软件发布后断言检查通常是关闭的。断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为true；如果表达式的值为false，那么系统会报告一个AssertionError。

Error和Exception有什么区别？

答：Error表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况；Exception表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

List、Set、Map是否继承自Collection接口？

答：List、Set 是，Map 不是。Map是键值对映射容器，与List和Set有明显的区别，而Set存储的零散的元素且不允许有重复元素（数学中的集合也是如此），List是线性结构的容器，适用于按数值索引访问元素的情形。

List、Map、Set三个接口存取元素时，各有什么特点？

答：List以特定索引来存取元素，可以有重复元素。Set不能存放重复元素（用对象的equals()方法来区分元素是否重复）。Map保存键值对（key-value pair）映射，映射关系可以是一对一或多对一。Set和Map容器都有基于哈希存储和排序树的两种实现版本，基于哈希存储的版本理论存取时间复杂度为O(1)，而基于排序树版本的实现在插入或删除元素时会按照元素或元素的键（key）构成排序树从而达到排序和去重的效果。

Thread类的sleep()方法和对象的wait()方法都可以让线程暂停执行，它们有什么区别？

答：sleep()方法（休眠）是线程类（Thread）的静态方法，调用此方法会让当前线程暂停执行指定的时间，将执行机会（CPU）让给其他线程，但是对象的锁依然保持，因此休眠时间结束后会自动恢复（线程回到就绪状态，请参考第66题中的线程状态转换图）。wait()是Object类的方法，调用对象的wait()方法导致当前线程放弃对象的锁（线程暂停执行），进入对象的等待池（wait pool），只有调用对象的notify()方法（或notifyAll()方法）时才能唤醒等待池中的线程进入等锁池（lock pool），如果线程重新获得对象的锁就可以进入就绪状态。

线程的sleep()方法和yield()方法有什么区别？

1、sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；

2、 线程执行sleep()方法后转入阻塞（blocked）状态，而执行yield()方法后转入就绪（ready）状态；

3、sleep()方法声明抛出InterruptedException，而yield()方法没有声明任何异常；

4、sleep()方法比yield()方法（跟操作系统CPU调度相关）具有更好的可移植性。

当一个线程进入一个对象的synchronized方法A之后，其它线程是否可进入此对象的synchronized方法B？

答：不能。其它线程只能访问该对象的非同步方法，同步方法则不能进入。因为非静态方法上的synchronized修饰符要求执行方法时要获得对象的锁，如果已经进入A方法说明对象锁已经被取走，那么试图进入B方法的线程就只能在等锁池（注意不是等待池哦）中等待对象的锁。

请说出与线程同步以及线程调度相关的方法。

wait()：使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；

sleep()：使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要处理InterruptedException异常；

notify()：唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且与优先级无关；

notityAll()：唤醒所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态。

编写多线程程序有几种实现方式？

答：Java 5以前实现多线程有两种实现方法：一种是继承Thread类；另一种是实现Runnable接口。两种方式都要通过重写run()方法来定义线程的行为，推荐使用后者，因为Java中的继承是单继承，一个类有一个父类，如果继承了Thread类就无法再继承其他类了，显然使用Runnable接口更为灵活。Java 5以后创建线程还有第三种方式：实现Callable接口，该接口中的call方法可以在线程执行结束时产生一个返回值。

启动一个线程是调用run()还是start()方法？

答：启动一个线程是调用start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由JVM 调度并执行，这并不意味着线程就会立即运行。run()方法是线程启动后要进行回调（callback）的方法。

什么是线程池？

Java 5+中的Executor接口定义一个执行线程的工具。它的子类型即线程池接口是ExecutorService。要配置一个线程池是比较复杂的，尤其是对于线程池的原理不是很清楚的情况下，因此在工具类Executors里面提供了一些静态工厂方法，生成一些常用的线程池，如下所示：

- newSingleThreadExecutor：创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。
- newFixedThreadPool：创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

- newCachedThreadPool：创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说JVM）能够创建的最大线程大小。
- newScheduledThreadPool：创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。
- newSingleThreadExecutor：创建一个单线程的线程池。此线程池支持定时以及周期性执行任务的需求。

简述synchronized 和java.util.concurrent.locks.Lock的异同？

答：Lock是Java 5以后引入的新的API，和关键字synchronized相比主要相同点：Lock 能完成synchronized所实现的所有功能；主要不同点：Lock有比synchronized更精确的线程语义和更好的性能，而且不强制性的要求一定要获得锁。synchronized会自动释放锁，而Lock一定要求程序员手工释放，并且最好在finally 块中释放（这是释放外部资源的最好的地方）。

Java中如何实现序列化，有什么意义？

答：序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决对象流读写操作时可能引发的问题（如果不进行序列化可能会存在数据乱序的问题）。

事务的ACID是指什么？

原子性(Atomic)：事务中各项操作，要么全做要么全不做，任何一项操作的失败都会导致整个事务的失败；

一致性(Consistent)：事务结束后系统状态是一致的；

隔离性(Isolated)：并发执行的事务彼此无法看到对方的中间状态；

持久性(Durable)：事务完成后所做的改动都会被持久化，即使发生灾难性的失败。通过日志和同步备份可以在故障发生后重建数据。

JAVA中的几种基本数据类型是什么，各自占用多少字节。

byte(1) short(2) int(4) long(8) float(4) double(8) char(2) boolean(1byte)

ArrayList和LinkedList有什么区别

1.ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。
（LinkedList是双向链表，有next也有previous）
2.对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。
3.对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据。

讲讲类的实例化顺序，比如父类静态数据，构造函数，字段，子类静态数据，构造函数，字段，当new的时候，他们的执行顺序

1. 父类静态成员和静态初始化块 ，按在代码中出现的顺序依次执行
2．子类静态成员和静态初始化块 ，按在代码中出现的顺序依次执行
3． 父类实例成员和实例初始化块 ，按在代码中出现的顺序依次执行
4． 父类构造方法
5． 子类实例成员和实例初始化块 ，按在代码中出现的顺序依次执行
6． 子类构造方法
结论：对象初始化的顺序，先静态方法，再构造方法，每个又是先基类后子类。

用过哪些Map类，都有什么区别，HashMap是线程安全的吗,并发下使用的Map是什么，他们内部原理分别是什么，比如存储方式，hashcode，扩容，默认容量等

hashmap(16)/treemap(11)/concurrentMap(分段16)

有没有有顺序的Map实现类，如果有，他们是怎么保证有序的。

TreeMap和LinkedHashmap都是有序的。（TreeMap默认是key升序，LinkedHashmap默认是数据插入顺序）
TreeMap是基于比较器Comparator来实现有序的

抽象类和接口的区别，类可以继承多个类么，接口可以继承多个接口么,类可以实现多个接口么。

不支持多个继承，支持多重继承，接口支持多继承，类可以实现多个接口。

继承和聚合的区别在哪

继承是扩展，聚合是包含

深拷贝与浅拷贝

浅拷贝只是复制了对象的引用地址，两个对象指向同一个内存地址，所以修改其中任意的值，另一个值都会随之变化，这就是浅拷贝（例：assign()）
深拷贝是将对象及值复制过来，两个对象修改其中任意的值另一个值不会改变，这就是深拷贝（例：JSON.parse()和JSON.stringify()，但是此方法无法复制函数类型）

请列出5个运行时异常
java.lang.NullPointerException
java.lang.ClassNotFoundException
java.lang.NoSuchMethodError
java.lang.IndexOutOfBoundsException
java.lang.NumberFormatException

答案是否定的。我们不能实现。为什么呢？我看很多网上解释是说双亲委托机制解决这个问题，其实不是非常的准确。因为双亲委托机制是可以打破的，你完全可以自己写一个classLoader来加载自己写的java.lang.String类，但是你会发现也不会加载成功，具体就是因为针对java.*开头的类，jvm的实现中已经保证了必须由bootstrp来加载。

在自己的代码中，如果创建一个java.lang.String类，这个类是否可以被类加载器加载？为什么
因加载某个类时，优先使用父类加载器加载需要使用的类。如果我们自定义了java.lang.String这个类， 加载该自定义的String类，该自定义String类使用的加载器是AppClassLoader，根据优先使用父类加载器原理， AppClassLoader加载器的父类为ExtClassLoader，所以这时加载String使用的类加载器是ExtClassLoader， 但是类加载器ExtClassLoader在jre/lib/ext目录下没有找到String.class类。然后使用ExtClassLoader父类的加载器BootStrap， 父类加载器BootStrap在JRE/lib目录的rt.jar找到了String.class，将其加载到内存中。这就是类加载器的委托机制

hashCode和equals方法的区别：

equal()相等的两个对象他们的hashCode()肯定相等，也就是用equal()对比是绝对可靠的。hashCode()相等的两个对象他们的equal()不一定相等，也就是hashCode()不是绝对可靠的。

因为重写的equal（ ）里一般比较的比较全面比较复杂，这样效率就比较低，而利用hashCode()进行对比，则只要生成一个hash值进行比较就可以了，效率很高，那么

hashCode()既然效率这么高为什么还要equal()呢？

因为hashCode()并不是完全可靠，有时候不同的对象他们生成的hashcode也会一样（生成hash值得公式可能存在的问题），所以hashCode()只能说是大部分时候可靠，并不是绝对可靠。

解决方式：当需要对比的时候，首先用hashCode()去对比，如果hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用equal()去再对比了）,如果hashCode()相同，此时再对比他们的equal()，如果equal()也相同，则表示这两个对象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性。

堆栈溢出的常见情况
栈溢出：递归调用没有终止条件，循环调用
堆溢出：创建过多对象，GC没有来得及回收

你们线上应用的JVM参数有哪些。
-Xms:设置堆的最小空间大小。不超过物理内存一半
-Xmx:设置堆的最大空间大小。一般与xms相同

g1和cms区别,吞吐量优先和响应优先的垃圾收集器选择。
https://www.cnblogs.com/rgever/p/9534857.html

Lock与Synchronized的区别
1.首先synchronized是java内置关键字，在jvm层面，Lock是个java类；
2.synchronized无法判断是否获取锁的状态，Lock可以判断是否获取到锁；
3.synchronized会自动释放锁(a 线程执行完同步代码会释放锁 ；b 线程执行过程中发生异常会释放锁)，Lock需在finally中手工释放锁（unlock()方法释放锁），否则容易造成线程死锁；
4.用synchronized关键字的两个线程1和线程2，如果当前线程1获得锁，线程2线程等待。如果线程1阻塞，线程2则会一直等待下去，而Lock锁就不一定会等待下去，如果尝试获取不到锁，线程可以不用一直等待就结束了；
5.synchronized的锁可重入、不可中断、非公平，而Lock锁可重入、可判断、可公平（两者皆可）
6.Lock锁适合大量同步的代码的同步问题，synchronized锁适合代码少量的同步问题。

在资源竞争不是很激烈的情况下，Synchronized的性能要优于ReetrantLock，但是在资源竞争很激烈的情况下，Synchronized的性能会下降几十倍，但是ReetrantLock的性能能维持常态；

线程池的关闭方式有几种，各自的区别是什么

线程池提供了两个关闭方法，shutdownNow和shuwdown方法。
shutdownNow方法的解释是：线程池拒接收新提交的任务，同时立马关闭线程池，线程池里的任务不再执行。
shutdown方法的解释是：线程池拒接收新提交的任务，同时等待线程池里的任务执行完毕后关闭线程池。

多线程如果线程挂住了怎么办

（1）通过调用sleep()方法使线程进入休眠状态，线程在指定时间内不会运行。
（2）通过调用wait()方法使线程挂起，直到线程得到了notify()和notifyAll()消息，线程才会进入“可执行”状态。

使用synchronized修饰静态方法和非静态方法有什么区别

synchronized是对类的当前实例进行加锁，防止其他线程同时访问该类的该实例的所有synchronized块，注意这里是“类的当前实例”，类的两个不同实例就没有这种约束了。
static synchronized恰好就是要控制类的所有实例的访问了，static synchronized是限制线程同时访问jvm中该类的所有实例同时访问对应的代码块。

简述ConcurrentLinkedQueue和LinkedBlockingQueue的用处和不同之处。

LinkedBlockingQueue实现是线程安全的，实现了先进先出等特性，是作为生产者消费者的首选，LinkedBlockingQueue 可以指定容量，也可以不指定，不指定的话，默认最大是Integer.MAX_VALUE，其中主要用到put和take方法，put方法在队列满的时候会阻塞直到有队列成员被消费，take方法在队列空的时候会阻塞，直到有队列成员被放进来。

ConcurrentLinkedQueue是一个适用于高并发场景下的队列，通过无锁的方式，实现了高并发状态下的高性能，通常ConcurrentLikedQueue性能好于BlockingQueue。它是一个基于连接节点的无界线程安全队列。该队列的元素遵循先进先出的原则。头是最先加入的，尾是最近加入的，该队列不允许null元素。

导致线程死锁的原因？怎么解除线程死锁。

多个线程同时被阻塞，它们中的一个或者全部都在等待某个资源被释放，而该资源又被其他线程锁定，从而导致每一个线程都得等其它线程释放其锁定的资源，造成了所有线程都无法正常结束。这是从网上其他文档看到的死锁产生的四个必要条件：

- 1、互斥使用，即当资源被一个线程使用(占有)时，别的线程不能使用
- 2、不可抢占，资源请求者不能强制从资源占有者手中夺取资源，资源只能由资源占有者主动释放。
- 3、请求和保持，即当资源请求者在请求其他的资源的同时保持对原有资源的占有。
- 4、循环等待，即存在一个等待队列：P1占有P2的资源，P2占有P3的资源，P3占有P1的资源。这样就形成了一个等待环路。

当上述四个条件都成立的时候，便形成死锁。当然，死锁的情况下如果打破上述任何一个条件，便可让死锁消失。下面用java代码来模拟一下死锁的产生。

开启多个线程，如果保证顺序执行，有哪几种实现方式，或者如何保证多个线程都执行完再拿到结果。

在子线程中通过join()方法指定顺序

在主线程中通过join()方法指定顺序

通过倒数计时器CountDownLatch实现

通过创建单一化线程池newSingleThreadExecutor()实现

https://blog.csdn.net/jqc874789596/article/details/100557300