Java

基础

Q：面向对象编程的四大特性及其含义?
Q：String、StringBuffer和StringBuilder的区别?
Q：String a=""和String a=new String("")的的关系和异同?
Q：Object的equal()和==的区别?
Q：装箱、拆箱什么含义?
Q：int和Integer的区别?
Q：遇见过哪些运行时异常？异常处理机制知道哪些?
Q：什么是反射，有什么作用和应用?
Q：什么是内部类？有什么作用？静态内部类和非静态内部类的区别?
Q：final、finally、finalize()分别表示什么含义
Q：重写和重载的区别?
Q：抽象类和接口的异同?
Q：为什么匿名内部类中使用局部变量要用final修饰?
Q：Object有哪些公有方法?

集合

Q：Java集合框架中有哪些类？都有什么特点
Q：集合、数组、泛型的关系，并比较
Q：ArrayList和LinkList的区别?
Q：ArrayList和Vector的区别?
Q：HashSet和TreeSet的区别?
Q：HashMap和Hashtable的区别?
Q：HashMap在put、get元素的过程？体现了什么数据结构?
Q：如何解决Hash冲突?
Q：如何保证HashMap线程安全？什么原理?
Q：HashMap是有序的吗？如何实现有序?
Q：HashMap是如何扩容的？如何避免扩容?
Q：hashcode()的作用，与equal()有什么区别?

并发

Q：同步和非同步、阻塞和非阻塞的概念
Q：Thread的join()有什么作用?
Q：线程的有哪些状态?
Q：什么是线程安全？保障线程安全有哪些手段?
Q：ReentrantLock和synchronized的区别?
Q：synchronized和volatile的区别?
Q：synchronized同步代码块还有同步方法本质上锁住的是谁？为什么?
Q：sleep()和wait()的区别?

JVM

Q：JVM内存是如何划分的?
Q：谈谈垃圾回收机制？为什么引用计数器判定对象是否回收不可行？知道哪些垃圾回收算法?
Q：Java中引用有几种类型？在Android中常用于什么情景?
Q：类加载的全过程是怎样的？什么是双亲委派模型?
Q：工作内存和主内存的关系？在Java内存模型有哪些可以保证并发过程的原子性、可见性和有序性的措施?
Q：JVM、Dalvik、ART的区别?
Q：Java中堆和栈的区别?

Java

基础

Q：面向对象编程的四大特性及其含义?

技术点：面向对象编程特点
思路：分条简述每个特性的含义
参考回答：

抽象：对现实世界的事物进行概括，抽象为在计算机虚拟世界中有意义的实体
封装：将某事物的属性和行为包装到对象中，构成一个不可分割的独立实体，数据被保护在抽象数据类型的内部，并且尽可能地隐藏内部的细节，只保留一些对外接口使之与外部发生联系
继承：子类继承父类，不仅可以有父类原有的方法和属性，也可以增加自己的或者重写父类的方法及属性
多态：允许不同类的对象对同一消息做出各自的响应

Q：String、StringBuffer和StringBuilder的区别?

技术点：String
参考回答：

String是字符串常量，而StringBuffer、StringBuilder都是字符串变量，即String对象一创建后不可更改，而后两者的对象是可更改的：
StringBuffer是线程安全的，而StringBuilder是非线程安全的，这是由于StringBuffer对方法加了同步锁或者对调用的方法加了同步锁
String更适用于少量的字符串操作的情况，StringBuilder适用于单线程下在字符缓冲区进行大量操作的情况，StringBuffer适用于多线程下在字符缓冲区进行大量操作的情况

Q：String a=""和String a=new String("")的的关系和异同?

技术点：String
参考回答：

通过String a=""直接赋值的方式得到的是一个字符串常量，存在于常量池；注意，相同内容的字符串在常量池中只有一个，即如果池已包含内容相等的字符串会返回池中的字符串，反之会将该字符串放入池中
通过new String("")创建的字符串不是常量是实例对象，会在堆内存开辟空间并存放数据，且每个实例对象都有自己的地址空间

引申：对于用String a=""和String a=new String("")两种方式定义的字符串，判断使用equals()、"=="比较结果是什么

Q：Object的equal()和==的区别?

技术点：equal()、==
参考回答：

equals()：是Object的公有方法，具体含义取决于如何重写，比如String的equals()比较的是两个字符串的内容是否相同
"==" ：对于基本数据类型来说，比较的是两个变量值是够是否相等，对于引用类型来说，比较的是两个对象的内存地址是否相同

引申：对于用String a=""和String a=new String("")两种方式定义的字符串，判断使用equals()、"=="比较结果是什么

Q：装箱、拆箱什么含义?

技术点：装箱、拆箱
参考回答：装箱就是自动将基本数据类型转换为包装器类型，拆箱就是自动将包装器类型转换为基本数据类型

Q：int和Integer的区别?

技术点：基本数据类型、引用类型
参考回答：

Integer是int的包装类，int则是java的一种基本数据类型
Integer变量必须实例化后才能使用，而int变量不需要
Integer实际是对象的引用，当new一个Integer时，实际上是生成一个指针指向此对象；而int则是直接存储数据值
Integer的默认值是null，int的默认值是0

Q：遇见过哪些运行时异常？异常处理机制知道哪些?

技术点：Java异常机制
思路：对Throwable异常进行分类说明每种异常的特点和常见问题，简述几种常见异常处理机制，详见Java基础之异常机制

参考回答：
（1） Throwable继承层次结构，可见分成两大类Error和Exception：

Error（错误）:指程序无法恢复的异常情况，表示运行应用程序中较严重的问题；发生于虚拟机自身、或者在虚拟机试图执行应用时，如Virtual MachineError（Java虚拟机运行错误）、NoClassDefFoundError（类定义错误）；属于不可查异常，即不强制程序员必须处理，即使不处理也不会出现语法错误。
Exception（异常）:指程序有可能恢复的异常情况，表示程序本身可以处理的异常。又分两大类：

RuntimeException（运行时异常）：由程序自身的问题导致产生的异常；如NullPointerException（空指针异常）、IndexOutOfBoundsException（下标越界异常）；属于不可查异常。
非运行时异常：由程序外部的问题引起的异常；除了RuntimeException以外的异常，如FileNotFoundException（文件不存在异常）；属于可查异常，即强制程序员必须进行处理，如果不进行处理则会出现语法错误。

Error		RuntimeException		除RuntimeException以外Exception	
异常类名	用途	异常类名	用途	异常类名	用途
LinkageError	动态链接失败	ArithmeticException	数学运算异常，比如除数为零的异常	ClassNotFoundException	指定类或接口不存在的异常
VirtualMachineError	虚拟机错误	IndexOutOfBoundsException	下标越界异常，比如集合、数组等	IllegalAccessException	非法访问异常
AWTError	AWT错误	ArrayIndexOutOfBoundsException	访问数组元素的下标越界异常	Ioexception	输入输出异常
		StringIndexOutOfBoundsException	字符串下标越界异常	FileNotFoundException	找不到指定文件的异常
		ClassCaseException	类强制转换异常	ProtocolException	网络协议异常
		NullpointerException	当程序试图访问一个空数组中的元素，或访问一个空对象中的方法或变量时产生的异常。	SocketException	Socket操作异常
				MalformedURLException	统一资源定位符（URL）的格式不正确

（2）常见的异常处理机制有：

捕捉异常：由系统自动抛出异常，即try捕获异常->catch处理异常->finally 最终处理
抛出异常：在方法中将异常对象显性地抛出，之后异常会沿着调用层次向上抛出，交由调用它的方法来处理。配合throws声明抛出的异常和throw抛出异常
自定义异常：继承Execption类或其子类

Q：什么是反射，有什么作用和应用?

技术点：反射
思路：简述反射的定义、功能和应用，详见Java基础之泛型&反射

参考回答：

含义：在运行状态中，对于任意一个类都能知道它的所有属性和方法，对于任何一个对象都能够调用它的任何一个方法和属性。

功能：动态性，体现在：在运行时判断任意一个类所具有的属性和方法； 在运行时判断任意一个对象所属的类；在运行时构造任意一个类的对象；在运行时调用任意一个对象的方法；生成动态代理
应用：反射&泛型

Q：什么是内部类？有什么作用？静态内部类和非静态内部类的区别?

技术点：内部类
思路：
参考回答：内部类就是定义在另外一个类里面的类。它隐藏在外部类中，封装性更强，不允许除外部类外的其他类访问它；但它可直接访问外部类的成员。静态内部类和非静态内部类的区别有：

静态内部类是指被声明为static的内部类，可不依赖外部类实例化；而非静态内部类需要通过生成外部类来间接生成。
静态内部类只能访问外部类的静态成员变量和静态方法，而非静态内部类由于持有对外部类的引用，可以访问外部类的所用成员

Q：final、finally、finalize()分别表示什么含义

技术点：final、finally、finalize()
参考回答：

final关键字表示不可更改，具体体现在：

final修饰的变量必须要初始化，且赋初值后不能再重新赋值

final修饰的方法不能被子类重写
final修饰的类不能被继承

finally：和try、catch成套使用进行异常处理，无论是否捕获或处理异常，finally块里的语句都会被执行，在以下4种特殊情况下，finally块才不会被执行：

在finally语句块中发生了异常
在前面的代码中用了System.exit()退出程序
程序所在的线程死亡
关闭CPU

finalize()：是Object中的方法，当垃圾回收器将回收对象从内存中清除出去之前会调用finalize()，但此时并不代表该回收对象一定会“死亡”，还有机会“逃脱”

Q：重写和重载的区别?

技术点：重写、重载
参考回答：重写表示子类重写父类的方法；重载表示有多个同名函数同时存在，区别在于有不同的参数个数或类型
引申：谈谈动态分派和静态分派

Q：抽象类和接口的异同?

技术点：抽象类、接口
参考回答：
使用上的区别：一个类只能继承一个抽象类却可以实现多个接口
设计上的区别：接口是对行为的抽象，无需有子类的前提，是自上而下的设计理念；抽象类是对类的抽象，建立于相似子类之上，是自下而上的设计理念

Q：为什么匿名内部类中使用局部变量要用final修饰?

技术点：匿名内部类
参考回答：一方面，由于方法中的局部变量的生命周期很短，一旦方法结束变量就要被销毁，为了保证在内部类中能找到外部局部变量，通过final关键字可得到一个外部变量的引用；另一方面，通过final关键字也不会在内部类去做修改该变量的值，保护了数据的一致性。

Q：Object有哪些公有方法?

技术点：Object
思路：列举常见的几个公有方法
参考回答：

equals()： 和==作用相似
hashCode()：用于哈希查找，重写了equals()一般都要重写该方法
getClass()： 获取Class对象
wait()：让当前线程进入等待状态，并释放它所持有的锁
notify()&notifyAll()： 唤醒一个（所有）正处于等待状态的线程
toString()：转换成字符串

引申：equals()和==的不同、在synchronized 同步代码块里wait()和notify()&notifyAll()如何配合、hashCode()和equals()的关系、获取Class对象还有什么方法

集合

Q：Java集合框架中有哪些类？都有什么特点

技术点：集合框架
思路：分条解释每种类的特点
参考回答：可将Java集合框架大致可分为Set、List、Queue 和Map四种体系

Set：代表无序、不可重复的集合，常见的类如HashSet、TreeSet
List：代表有序、可重复的集合，常见的类如动态数组ArrayList、双向链表LinkedList、可变数组Vector
Map：代表具有映射关系的集合，常见的类如HashMap、LinkedHashMap、TreeMap
Queue：代表一种队列集合

Q：集合、数组、泛型的关系，并比较

技术点：集合、数组、泛型
参考回答：
（1）集合和数组的区别：

数组元素可以是基本类型，也可以是对象；数组长度限定；数组只能存储一种类型的数据元素
集合元素只能是对象；集合长度可变；集合可存储不同种的数据元素

（2）泛型相比与集合的好处在于它安全简单。具体体现在提供编译时的强类型检查，而不用等到运行；可避免类类型强制转换

Q：ArrayList和LinkList的区别?

技术点：List对比
参考回答：
ArrayList的底层结构是数组，可用索引实现快速查找；是动态数组，相比于数组容量可实现动态增长
LinkedList底层结构是链表，增删速度快；是一个双向循环链表，也可以被当作堆栈、队列或双端队列

Q：ArrayList和Vector的区别?

技术点：List对比
参考回答：

ArrayList非线程安全，建议在单线程中才使用ArrayList，而在多线程中可以选择Vector或者CopyOnWriteArrayList；默认初始容量为10，每次扩容为原来的1.5倍

Vector使用了synchronized关键字，是线程安全的，比ArrayList开销更大，访问更慢；默认初始容量为10，默认每次扩容为原来的2倍，可通过capacityIncrement属性设置

Q：HashSet和TreeSet的区别?

技术点：Set对比
参考回答：

HashSet不能保证元素的排列顺序；使用Hash算法来存储集合中的元素，有良好的存取和查找性能；通过equal()判断两个元素是否相等，并两个元素的hashCode()返回值也相等

TreeSet是SortedSet接口的实现类，根据元素实际值的大小进行排序；采用红黑树的数据结构来存储集合元素；支持两种排序方法：自然排序（默认情况）和定制排序。前者通过实现Comparable接口中的compareTo()比较两个元素之间大小关系，然后按升序排列；后者通过实现Comparator接口中的compare()比较两个元素之间大小关系，实现定制排列

Q：HashMap和Hashtable的区别?

技术点：Map对比
参考回答：

HashMap基于AbstractMap类，实现了Map、Cloneable（能被克隆）、Serializable（支持序列化）接口； 非线程安全；允许存在一个为null的key和任意个为null的value；采用链表散列的数据结构，即数组和链表的结合；初始容量为16，填充因子默认为0.75，扩容时是当前容量翻倍，即2capacity

Hashtable基于Map接口和Dictionary类；线程安全，开销比HashMap大，如果多线程访问一个Map对象，使用Hashtable更好；不允许使用null作为key和value；底层基于哈希表结构；初始容量为11，填充因子默认为0.75，扩容时是容量翻倍+1，即2capacity+1

Q：HashMap在put、get元素的过程？体现了什么数据结构?

技术点：HashMap
参考回答：

向Hashmap中put元素时，首先判断key是否为空，为空则直接调用putForNullKey()，不为空则计算key的hash值得到该元素在数组中的下标值；如果数组在该位置处没有元素，就直接保存；如果有，还要比较是否存在相同的key，存在的话就覆盖原来key的value，否则将该元素保存在链头，先保存的在链尾。
从Hashmap中get元素时，计算key的hash值找到在数组中的对应的下标值，返回该key对应的value即可，如果有冲突就遍历该位置链表寻找key相同的元素并返回对应的value
由此可看出HashMap采用链表散列的数据结构，即数组和链表的结合，在Java8后又结合了红黑树，当链表元素超过8个将链表转换为红黑树

Q：如何解决Hash冲突?

技术点：Hash冲突
参考回答：

开放定址法：常见的线性探测方式，在冲突发生时，顺序查看表中下一单元，直到找出一个空单元或查遍全表
链地址法：将有冲突数组位置生出链表
建立公共溢出区：将哈希表分为基本表和溢出表两部分，和基本表发生冲突的元素一律填入溢出表
再哈希法：构造多个不同的哈希函数，有冲突使用下一个哈希函数计算hash值

Q：如何保证HashMap线程安全？什么原理？

技术点：ConcurrentHashMap
思路：这里回答一种办法，使用ConcurrentHashMap
参考回答：ConcurrentHashMap是线程安全的HashMap，它采取锁分段技术，将数据分成一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据的时候，其他段的数据也能被其他线程访问。在JDK1.8中对ConcurrentHashmap做了两个改进：

取消segments字段，直接采用transient volatile HashEntry<K,V>[] table保存数据，将数组元素作为锁，对每一行数据进行加锁，可减少并发冲突的概率
数据结构由“数组＋单向链表”变为“数组＋单向链表＋红黑树”，使得查询的时间复杂度可以降低到O(logN)，改进一定的性能。

引申：LinkHashMap线程安全的底层实现

Q：HashMap是有序的吗？如何实现有序？

技术点：LinkHashMap
思路：这里回答一种办法，使用LinkedHashMap
参考回答：HashMap是无序的，而LinkedHashMap是有序的HashMap，默认为插入顺序，还可以是访问顺序，基本原理是其内部通过Entry维护了一个双向链表，负责维护Map的迭代顺序
引申：LinkHashMap有序的底层实现

Q：HashMap是如何扩容的？如何避免扩容？

技术点：HashMap
参考回答：

HashMap几个默认值，初始容量为16、填充因子默认为0.75、扩容时容量翻倍。也就是说当HashMap中元素个数超过16*0.75=12时会把数组的大小扩展为2*16=32，然后重新计算每个元素在数组中的位置
由于每次扩容还需要重新计算元素Hash值，损耗性能，所以建议在使用HashMap时，最好先估算Map的大小，设置初始值，避免频繁扩容

Q：hashcode()的作用，与equal()有什么区别？

技术点：Hash值
参考回答：hashCode()用于计算对象的Hash值，确认对象在散列存储结构中的存储地址。和equal()的区别：

equals()比较两个对象的地址值是否相等 ；hashCode()得到的是对象的存储位置，可能不同对象会得到相同值
有两个对象，若equals()相等，则hashcode()一定相等；hashcode()不等，则equals()一定不相等；hashcode()相等，equals()可能相等、可能不等
使用equals()比较两个对象是否相等效率较低，最快办法是先用hashCode()比较，如果hashCode()不相等，则这两个对象肯定不相等；如果hashCode()相等，此时再用equal()比较，如果equal()也相等，则这两个对象的确相等，反之

并发

Q：同步和非同步、阻塞和非阻塞的概念

技术点：同步、阻塞
参考回答：

同步和异步体现的是消息的通知机制：所谓同步，方法A调用方法B后必须等到方法B返回结果才能继续后面的操作；所谓异步，方法A调用方法B后可让方法B在调用结束后通过回调等方式通知方法A
阻塞和非阻塞侧重于等待消息时的状态：所谓阻塞，就是在结果返回之前让当前线程挂起；所谓非阻塞，就是在等待时可做其他事情，通过轮询去询问是否已返回结果

Q：Thread的join()有什么作用?

技术点：线程相关方法
参考回答：Thread的join()的含义是等待该线程终止，即将挂起调用线程的执行，直到被调用的对象完成它的执行。比如存在两个线程t1和t2，下述代码表示先启动t1，直到t1的任务结束，才轮到t2启动。

```
t1.start();
t1.join();
t2.start();
```

Q：线程的有哪些状态?

技术点：线程状态
思路：可分条解释每种状态的特点以及如何转换。详见要点提炼| 理解JVM之内存模型&线程

参考回答：在任意一个时间点，一个线程只能有且只有其中的一种状态：

新建（New）：线程创建后尚未启动

运行（Runable）：包括正在执行（Running）和等待着CPU为它分配执行时间（Ready）两种

无限期等待（Waiting）：该线程不会被分配CPU执行时间，要等待被其他线程显式地唤醒。以下方法会让线程陷入无限期等待状态：

没有设置Timeout参数的Object.wait()

没有设置Timeout参数的Thread.join()

LockSupport.park()

限期等待（Timed Waiting）：该线程不会被分配CPU执行时间，但在一定时间后会被系统自动唤醒。以下方法会让线程进入限期等待状态：

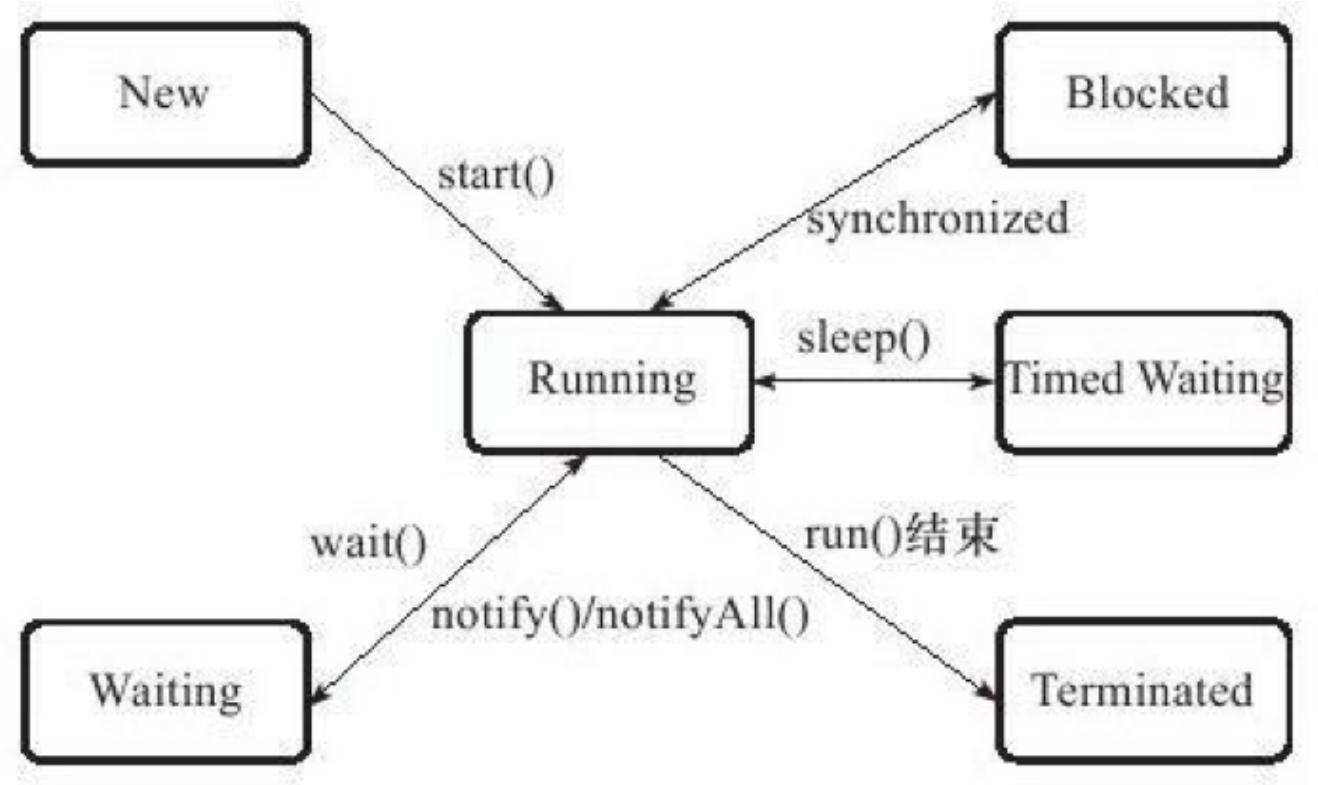
Thread.sleep()
设置了Timeout参数的Object.wai()

设置了Timeout参数的Thread.join()

LockSupport.parkNanos()
LockSupport.parkUntil()

阻塞（Blocked）：线程被阻塞。和等待状态不同的是，阻塞状态表示在等待获取到一个排他锁，在另外一个线程放弃这个锁的时候发生；而等待状态表示在等待一段时间或者唤醒动作的发生，在程序等待进入同步区域的时候发生。

结束（Terminated）：线程已经结束执行



Q：什么是线程安全？保障线程安全有哪些手段?

技术点：线程安全
思路：详见要点提炼| 理解JVM之线程安全&锁优化

参考回答：线程安全就是当多个线程访问一个对象时，如果不用考虑这些线程在运行时环境下的调度和交替执行，也不需要进行额外的同步，或者在调用方进行任何其他的协调操作，调用这个对象的行为都可以获得正确的结果，那这个对象是线程安全的。保证线程安全可从多线程三特性出发：

原子性（Atomicity）：单个或多个操作是要么全部执行，要么都不执行

Lock：保证同时只有一个线程能拿到锁，并执行申请锁和释放锁的代码
synchronized：对线程加独占锁，被它修饰的类/方法/变量只允许一个线程访问

可见性（Visibility）：当一个线程修改了共享变量的值，其他线程能够立即得知这个修改

volatile：保证新值能立即同步到主内存，且每次使用前立即从主内存刷新；
synchronized：在释放锁之前会将工作内存新值更新到主存中

有序性（Ordering）：程序代码按照指令顺序执行

volatile： 本身就包含了禁止指令重排序的语义
synchronized：保证一个变量在同一个时刻只允许一条线程对其进行lock操作，使得持有同一个锁的两个同步块只能串行地进入

Q：ReentrantLock和synchronized的区别?

技术点：线程安全（ReentrantLock、synchronized）
思路：详见要点提炼| 理解JVM之线程安全&锁优化

参考回答： ReentrantLock与synchronized的不同在于ReentrantLock：

等待可中断：当持有锁的线程长期不释放锁的时候，正在等待的线程可以选择放弃等待，改为处理其他事情。

公平锁：多个线程在等待同一个锁时，必须按照申请锁的时间顺序来依次获得锁。而synchronized是非公平的，即在锁被释放时，任何一个等待锁的线程都有机会获得锁。ReentrantLock默认情况下也是非公平的，但可以通过带布尔值的构造函数改用公平锁。

锁绑定多个条件：一个ReentrantLock对象可以通过多次调用newCondition()同时绑定多个Condition对象。而在synchronized中，锁对象wait()和notify()或notifyAl()只能实现一个隐含的条件，若要和多于一个的条件关联不得不额外地添加一个锁。

Q：synchronized和volatile的区别?

技术点：线程安全（synchronized、volatile）
参考回答：

synchronized能保证操作的原子性，而volatile不可以，假设线程A和线程B同时读取到变量a值，A修改a后将值更新到主内存，同时B也修改a值会覆盖A的修改操作
synchronized可修饰变量、方法和类，而volatile只能修饰变量
synchronized可能会造成线程阻塞，而volatile不会造成线程的阻塞

Q：synchronized同步代码块还有同步方法本质上锁住的是谁？为什么?

技术点：线程安全（synchronized）
参考回答：本质上锁住的是对象。在java虚拟机中，每个对象和类在逻辑上都和一个监视器相关联，synchronized本质上是对一个对象监视器的获取。当执行同步代码块或同步方法时，执行方法的线程必须先获得该对象的监视器，才能进入同步代码块或同步方法；而没有获取到的线程将会进入阻塞队列，直到成功获取对象监视器的线程执行结束并释放锁后，才会唤醒阻塞队列的线程，使其重新尝试对对象监视器的获取。

Q：sleep()和wait()的区别?

技术点：sleep()、wait()
参考回答：

sleep()来自Thread类；wait()来自Object类
sleep()用于线程控制自身流程；而wait()用于线程间通信，配合notify()/notifyAll()在同步代码块或同步方法里使用
sleep()的线程不会释放对象锁；wait()会释放对象锁进入等待状态，使得其他线程能使用同步代码块或同步方法

JVM

Q：JVM内存是如何划分的?

技术点：JVM内存管理
思路：分条解释每部分内存的作用，详见要点提炼| 理解JVM之内存管理

参考回答：JVM会用一段空间来存储执行程序期间需要用到的数据和相关信息，这段空间就是运行时数据区（Runtime Data Area），也就是常说的JVM内存。JVM会将它所管理的内存划分为线程私有数据区和线程共享数据区两大类：

线程私有数据区包含：

程序计数器：是当前线程所执行的字节码的行号指示器
虚拟机栈：是Java方法执行的内存模型
本地方法栈：是虚拟机使用到的Native方法服务

线程共享数据区包含：

Java堆：用于存放几乎所有的对象实例和数组；是垃圾收集器管理的主要区域，也被称做“GC堆”；是Java虚拟机所管理的内存中最大的一块
方法区：用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据；Class文件中除了有类的版本、字段、方法、接口等描述信息外，还有一项信息是常量池（Constant Pool Table），用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后进入方法区的运行时常量池中存放

Q：谈谈垃圾回收机制？为什么引用计数器判定对象是否回收不可行？知道哪些垃圾回收算法?

技术点：垃圾回收机制
思路：从如何判定对象可回收、如果回收具体算法这两方面展开谈垃圾回收机制，详见要点提炼| 理解JVM之GC

参考回答：
（1）判定对象可回收有两种方法：

引用计数算法：给对象中添加一个引用计数器，每当有一个地方引用它时，计数器值就加1；当引用失效时，计数器值就减1；任何时刻计数器为0的对象就是不可能再被使用的。然而在主流的Java虚拟机里未选用引用计数算法来管理内存，主要原因是它难以解决对象之间相互循环引用的问题，所以出现了另一种对象存活判定算法。

可达性分析法：通过一系列被称为『GC Roots』的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链，当一个对象到GC Roots没有任何引用链相连时，则证明此对象是不可用的。其中可作为GC Roots的对象：虚拟机栈中引用的对象，主要是指栈帧中的本地变量、本地方法栈中Native方法引用的对象、方法区中类静态属性引用的对象、方法区中常量引用的对象

（2）回收算法有以下四种：

分代收集算法：是当前商业虚拟机都采用的一种算法，根据对象存活周期的不同，将Java堆划分为新生代和老年代，并根据各个年代的特点采用最适当的收集算法。

新生代：大批对象死去，只有少量存活。使用『复制算法』，只需复制少量存活对象即可。

复制算法：把可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用尽后，把还存活着的对象『复制』到另外一块上面，再将这一块内存空间一次清理掉。

老年代：对象存活率高。使用『标记-清理算法』或者『标记-整理算法』，只需标记较少的回收对象即可。

标记-清除算法：首先『标记』出所有需要回收的对象，然后统一『清除』所有被标记的对象。

标记-整理算法：首先『标记』出所有需要回收的对象，然后进行『整理』，使得存活的对象都向一端移动，最后直接清理掉端边界以外的内存。

引申：谈谈每种回收算法的优缺点

Q：Java中引用有几种类型？在Android中常用于什么情景？

技术点：Java引用类型
思路：分条解释每种类型的特点和适用场景，详见要点提炼| 理解JVM之GC

参考回答：引用的四种类型

强引用（StrongReference）：具有强引用的对象不会被GC；即便内存空间不足，JVM宁愿抛出OutOfMemoryError使程序异常终止，也不会随意回收具有强引用的对象。

软引用（SoftReference）：只具有软引用的对象，会在内存空间不足的时候被GC；软引用常用来实现内存敏感的高速缓存。

弱引用（WeakReference）：只被弱引用关联的对象，无论当前内存是否足够都会被GC；强度比软引用更弱，常用于描述非必需对象；常用于解决内存泄漏的问题

虚引用（PhantomReference）：仅持有虚引用的对象，在任何时候都可能被GC；常用于跟踪对象被GC回收的活动；必须和引用队列 （ReferenceQueue）联合使用，当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之关联的引用队列中。

Q：类加载的全过程是怎样的？什么是双亲委派模型？

技术点：类加载机制、双亲委派模型
思路：类加载机制的含义以及每个阶段的作用，在解释双亲委派模型之前需要先理解类加载器，详见要点提炼| 理解JVM之类加载机制

参考回答：
（1）类加载机制：是虚拟机把描述类的数据从Class文件加载到内存，并对数据进行校验、转换解析和初始化，最终形成可被虚拟机直接使用的Java类型的过程。另外，类型的加载、连接和初始化过程都是在程序运行期完成的，从而通过牺牲一些性能开销来换取Java程序的高度灵活性。下面介绍类加载每个阶段的任务：

加载（Loading）：通过类的全限定名来获取定义此类的二进制字节流；将该二进制字节流所代表的静态存储结构转化为方法区的运行时数据结构，该数据存储数据结构由虚拟机实现自行定义；在内存中生成一个代表这个类的java.lang.Class对象，它将作为程序访问方法区中的这些类型数据的外部接口

验证（Verification）：确保Class文件的字节流中包含的信息符合当前虚拟机的要求，包括文件格式验证、元数据验证、字节码验证和符号引用验证

准备（Preparation）：为类变量分配内存，因为这里的变量是由方法区分配内存的，所以仅包括类变量而不包括实例变量，后者将会在对象实例化时随着对象一起分配在Java堆中；设置类变量初始值，通常情况下零值

解析（Resolution）：虚拟机将常量池内的符号引用替换为直接引用的过程

初始化（Initialization）：是类加载过程的最后一步，会开始真正执行类中定义的Java字节码。而之前的类加载过程中，除了在『加载』阶段用户应用程序可通过自定义类加载器参与之外，其余阶段均由虚拟机主导和控制

（2）类加载器：不仅用于加载类，还和这个类本身一起作为在JVM中的唯一标识。常见类加载器类型有：

启动类加载器：是虚拟机自身的一部分
扩展类加载器、应用程序类加载器、自定义类加载器：独立于虚拟机外部

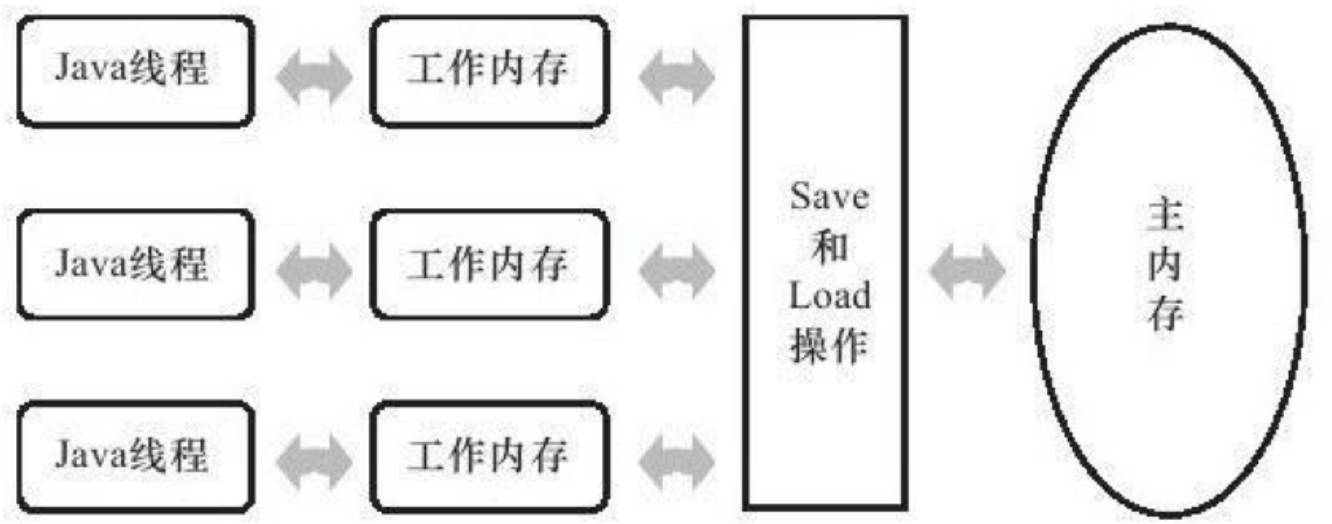
（3）双亲委派模型：表示类加载器之间的层次关系。

前提：除了顶层启动类加载器外，其余类加载器都应当有自己的父类加载器，且它们之间关系一般不会以继承（Inheritance）关系来实现，而是通过组合（Composition）关系来复用父加载器的代码。

工作过程：若一个类加载器收到了类加载的请求，它先会把这个请求委派给父类加载器，并向上传递，最终请求都传送到顶层的启动类加载器中。只有当父加载器反馈自己无法完成这个加载请求时，子加载器才会尝试自己去加载。

Q：工作内存和主内存的关系？在Java内存模型有哪些可以保证并发过程的原子性、可见性和有序性的措施？

技术点：JVM内存模型、线程安全
思路：理解Java内存模型的结构、详见要点提炼| 理解JVM之内存模型&线程
参考回答：Java内存模型就是通过定义程序中各个变量的访问规则，即在虚拟机中将变量存储到内存和从内存中取出变量这样的底层细节。
　　模型结构如图：



其中 ，主内存（Main Memory）是所有变量的存储位置，每条线程还有自己的工作内存，用于保存被该线程使用到的变量的主内存副本拷贝。为了获取更好的运行速度，虚拟机可能会让工作内存优先存储于寄存器和高速缓存中。
保证并发过程的原子性、可见性和有序性的措施：

原子性（Atomicity）：一个操作要么都执行要么都不执行。

可直接保证的原子性变量操作有：read、load、assign、use、store和write，因此可认为基本数据类型的访问读写是具备原子性的。
若需要保证更大范围的原子性，可通过更高层次的字节码指令monitorenter和monitorexit来隐式地使用lock和unlock这两个操作，反映到Java代码中就是同步代码块synchronized关键字。

可见性（Visibility）：当一个线程修改了共享变量的值，其他线程能够立即得知这个修改。

通过在变量修改后将新值同步回主内存，在变量读取前从主内存刷新变量值这种依赖主内存作为传递媒介的方式来实现。
提供三个关键字保证可见性：volatile能保证新值能立即同步到主内存，且每次使用前立即从主内存刷新；
synchronized对一个变量执行unlock操作之前可以先把此变量同步回主内存中；被final修饰的字段在构造器中一旦初始化完成且构造器没有把this的引用传递出去，就可以在其他线程中就能看见final字段的值。

有序性（Ordering）：程序代码按照指令顺序执行。

如果在本线程内观察，所有的操作都是有序的，指“线程内表现为串行的语义”；如果在一个线程中观察另一个线程，所有的操作都是无序的，指“指令重排序”现象和“工作内存与主内存同步延迟”现象。
提供两个关键字保证有序性：volatile 本身就包含了禁止指令重排序的语义；synchronized保证一个变量在同一个时刻只允许一条线程对其进行lock操作，使得持有同一个锁的两个同步块只能串行地进入。

Q：JVM、Dalvik、ART的区别?

技术点：虚拟机对比
思路：分别谈谈JVM和Dalvik、Dalvik和ART的区别，详见Jvm、Dalvik和Art的区别

参考回答：

Dalvik ：是Google公司自己设计用于Android平台的Java虚拟机，不是Java虚拟机，没有遵循Java虚拟机规范，具体区别如下图：

Java虚拟机	Dalvik虚拟机
java虚拟机基于栈，基于栈的机器必须使用指令来载入和操作栈上数据	Dalvik虚拟机基于寄存器
java虚拟机运行的是java字节码。（java类会被编译成一个或多个字节码.class文件，打包到.jar文件中，java虚拟机从相应的.class文件和.jar获取相应的字节码）	Dalvik运行的是自己专属的.dex字节码格式。（java类被编译成.class文件后，会通过一个dx工具将所有的.class文件转换成一个.dex文件，然后dalvik虚拟机会从其中读取指令和数据）
-	一个应用对应一个Diavik虚拟机实例，独立运行
JVM在运行的时候为每一个类装载字节码	Dalvik程序只包含一个.dex文件，这个文件包含了程序中所有的类

ART：代替Dalvik，应用无需每次运行都要先编译，而是在安装时就预编译字节码到机器语言，提升运行时效率；预先编译也使得ART占用空间比Dalvik大，即用空间换时间；由于减少运行时重复编译，可明显改善电池续航，降低了能耗。

Q：Java中堆和栈的区别?

技术点：内存管理
思路：从存放数据和内存回收角度出发
参考回答： 在java中，堆和栈都是内存中存放数据的地方，具题区别是：

栈内存：主要用来存放基本数据类型和局部变量；当在代码块定义一个变量时会在栈中为这个变量分配内存空间，当超过变量的作用域后这块空间就会被自动释放掉。
堆内存：用来存放运行时创建的对象，比如通过new关键字创建出来的对象和数组；需要由Java虚拟机的自动垃圾回收器来管理。