Java 语言进阶与 Android 技术相关面试题

Java 相关知识全面分析

1.什么是内部类？内部类的作用是什么？

内部类是定义在另一个类里面的类，与之相对应，包含内部类的类被称为外部类，

内部类的作用有：（1）内部类提供了更好的封装，可以把内部类隐藏在外部类之内，不允许同一包中的其他类访问，（2）内部类的方法可以直接访问外部类的所有数据，包括私有的数据，（3）内部类的种类：成员内部类、静态内部类、方法内部类、匿名内部类

2.进程与线程的区别

进程是 CPU 资源分配的最小单位，而线程是 CPU 调度的最小单位，进程之间不能共享资源，而线程共享所在进程的地址空间和其他资源，一个进程内可以拥有多个线程，进程可以开启进程、也可以开启线程，一个线程只能属于一个进程，线程可直接使用同进程的资源，线程依赖于进程而存在。

3.静态属性和静态方法是否可以被继承？是否可以被重写？

可以继承，但不可以被重写，而是被隐藏，如果子类里面定义了静态方法或属性，那么这时候父类的静态方法或属性称之为隐藏。

4.成员内部类、静态内部类、局部内部类、和匿名内部类的理解

Java 中内部类主要分为成员内部类、局部内部类（嵌套在方法和作用域中）、匿名内部类（无构造方法）、静态内部类（由 static 修饰的类、不能使用

任何外围类的非 static 成员变量和方法、不依赖外围类），每个内部类都能独立的继承一个（接口的）实现，所以无论外围类是否已经继承了某个（接口的）实现，对于内部类均无影响，因为 Java 支持实现多个接口，而不支持多继承，我们可以使用内部类提供的、可以继承多个具体的或抽象的类的能力来解决使用接口难以解决的问题，接口只是解决了部分问题，而内部类使得多继承的解决方案变得更加完整。

5.Java 的垃圾回收机制及其在何时会被触发

内存回收机制：就是释放掉在内存中已经没有用的对象，要判断怎样的对象是没用的，有两种方法：（1）采用标记数的方法，在给内存中的对象打上标记，对象被引用一次，计数加一，引用被释放，计数就减一，当这个计数为零时，这个对象就可以被回收，但是，此种方法，对于循环引用的对象是无法识别出来并加以回收的，（2）采用根搜索的方法，从一个根出发，搜索所有的可达对象，则剩下的对象就是可被回收的，垃圾回收是在虚拟机空闲的时候或者内存紧张的时候执行的，什么时候回收并不是由程序员控制的，可达与不可达的概念：分配对象使用 new 关键字，释放对象时，只需将对象的引用赋值为 null，让程序不能够在访问到这个对象，则称该对象不可达。

在以下情况中垃圾回收机制会被触发：

（1）所有实例都没有活动线程访问 ；（2）没有其他任何实例访问的循环引用实例；（3）Java 中有不同的引用类型。判断实例是否符合垃圾收集的条件都依赖于它的引用类型。

6.Java 中的代理是什么？静态代理和动态代理的区别是什么？

代理模式:在某些情况下，一个用户不想或不能直接引用一个对象，此时可以通过一个称之为“代理”的第三者来实现间接引用，代理对象可以在客户端和目标对象之间起中介的作用，并且可以通过中介对象去掉用户不能看到的内

容和服务，或者添加用户需要的额外服务。

静态代理：即在程序运行前代理类就已经存在，也就是编写代码的时候已经将代理类的代码写好。

动态代理：在程序运行时，通过反射机制动态创建代理类。

7.Java 中实现多态的机制是什么？

方法的重写 Overriding 和重载 Overloading 是 Java 多态性的不同表现

重写 Overriding 是父类与子类之间多态的一种表现；

重载 Overloading 是同一个类中多态性的一种表现；

8.Java 中反射的相关理解

Java 的反射机制是在运行状态中，对于任意一个类，都能够获取到这个类的所有属性和方法，对于任意一个对象，能够调用它的任意一个方法和属性，包括私有的方法和属性，这种动态地获取信息以及动态的调用对象的方法的功能就称之为 Java 的反射机制。

从对象出发，通过反射（.class 类）可以获取到类的完整信息，（类名、class 类型、所在包、具有的所有方法 Method[]类型、某个方法的完整信息，包括修饰符、返回值类型、异常、参数类型、所有属性 Field[ ]、某个属性的完整信息，构造器 Constructors,调用类的属性或方法。

9.Java 中注解的相关理解

Java 注解，英文名为 Annotation，一种代码级别的说明，是 JDK1.5 及以后版本引入的一个特性，与类、接口、枚举在同一个层次，他可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明、

注释、作用分类；注解是 Java 提供的一种元程序中元素关联任何信息和任何元数据（metadata）的途径和方法，Annotion(注解)是一个接口，程序可以通过反射来获取指定程序元素的 Annotion 对象，然后通过 Annotion 对象来获取注解里面的元数据

注解的一般格式为：[修饰符]@interface[名称]{元素}，元素是无方法体的方法声明，可以有默认值。

注解的作用：

编写文档：通过代码里标识的元数据生成文档[生成文档 doc]

代码分析：通过代码里的元数据对代码进行分析[使用反射]

编译检查：通过代码里标识的元数据让编译器能够实现基本的编译检查[Override]

10.对 Java 中 String 类的理解：

通过对 String 类的源代码分析可知，（1）String 类是 final 类，即意味着 String 类不能被继承，并且他的成员方法都默认为 final 方法；（2）String 类在源代码中实际上是通过 char[ ]数组来保存字符串的；（3）String 对象一旦创建就是固定不便的，对 String 对象的任何 change 操作都会生成新的对象。

11.对 Java 中字符串常量池的理解

字符串的分配和其他对象分配一样，是需要消耗高昂的时间和空间成本的，而且字符串在程序中使用得非常多，JVM 为了提高性能和减少内存的开销，在实例化字符串的时候会进行一些优化；每当我们创建字符串常量时，JVM 首先会检查字符串常量池，如果该字符床已经存在于常量池中，那么就直接返回常量池中的实例引用，如果该字符串不存在，就会实例化该字符串，并将其放在常量池中，由于字符串的不可变性，我们可以十分肯定常量池中一定

不存在两个相同的字符串，Java 中的常量池实际上分为两种形态：静态常量池和运行时常量池

静态常量池：即*.class 文件的常量池，.class 文件的常量池不仅仅包括字符串（数字）字面量，还包含类、方法的信息，占用 class 文件的绝大部分空间。

运行时常量池：常量存入到内存中，并保存在方法区中，我们常说的常量池，就是指方法区中的运行时常量池。

12.Java 中为什么 String 类要设计成不可变的

在 java 中将 String 类设计成不可变的是综合考虑到各种因素的结果，需要综合内存、同步、数据结构、以及安全等方面的考虑。

字符串常量池的需要：字符串常量池是 Java 堆内存中一个特殊的存储区域，当创建一个 String 对象时，假如此字符串值已经存在于常量池中，则不会创建一个新的对象，而是引用已经存在的对象，假若字符串对象允许改变，那么将会导致各种逻辑的错误，比如改变一个引用的字符串将会导致另一个引用出现脏数据。

允许 String 对象缓存 HashCode：Java 中 String 对象的哈希码被频繁的使用，比如在 HashMap 等容器中，字符串不变性保证了哈希码的唯一性，因此可以放心地进行缓存，这也是一种性能优化的手段，意味着不必每次都去计算新的哈希码。

安全性：String 被许多 Java 类库用来当作参数，如：网络连接（network connection）、打开文件（opening files）等等，如果 String 不是不可变的，网络连接、打开文件将会被改变——这将导致一系列的安全威胁，操作的方法本以为连接上一台机器，其实不是，优于反射的参数都是字符串，同样也会引起一系列的安全问题。

13.Java 中 Hash 码（哈希码）的理解

在 Java 中，哈希码代表了对象的一种特征，例如我们判断某两个字符串是否==，如果其哈希码相等，则这两个字符串是相等的，其次，哈希码是一种数据结构的算法，常见的哈希码的算法有：

Object 类的 HashCode，返回对象的内存地址经过处理后的结构，由于每个对象的内存地址都不一样，所以哈希码也不一样。

String 类的 HashCode，根据 String 类包含的字符串的内容，根据一种特殊的算法返回哈希码，只要字符串的内容相同，返回的哈希码也相同。

Integer 类：返回的哈希码就是 integer 对象里所包含的那个整数的数值。例如

Integer i1=new Integer(100) i1.hashCode 的值就是 100，由此可见两个一样大小的 Integer 对象返回的哈希码也一样。

14.Java 常用集合 List 与 Set，以及 Map 的区别

Java 中的集合主要分为三种类型：Set(集)、List(列表)、Map(映射)；

数组是大小固定的，并且同一个数组只能存放类型一样的数据（基本类型/引用类型），而 Java 集合是可以存储和操作数目不固定的一组数据，所有的 Java 集合都位于 java.util 包中，Java 集合只能存放引用类型的数据，不能存放基本数据类型。

Collection 是最基本的集合接口，声明了适用于 Java 集合（只包括 Set 和 List）的通用方法，Set 和 List 都继承了 Collection 接口。

Set 是最简单的一种集合，集合中的对象不按特定的方式排序，并且没有重复对象，Set 接口主要实现了两种实现类

TreeSet：TreeSet 类实现了 SortedSet 接口，能够对集合中的对象进行排序；

HashSet：HashSet 类按照哈希算法来存取集合中的对象，存取速度比较快；

Set 具有与 Collection 完全一样的接口，因此没有任何额外的功能，实际上 Set 就是 Collection,只是行为不同（这是继承与多态思想的典型应用，表现不同的行为），Set 不保存重复的元素，Set 接口不保证维护元素的次序。

（2） List 列表的特征是其他元素以线性表的方式存储，集合中可以存放重复的对象，其接口主要实现类：

ArrayList( ):代表长度可以改变的数组，可以对元素进行随机的访问，向 ArrayList（ ）中插入与删除元素的速度慢。

LinkedList( ):在实现类中采用链表数据结构，插入和删除的速度快，但访问的速度慢。

对于 List 的随机访问来说，就是只是随机来检索位于特定位置的元素，List 的 get(int index)方法返回集合中由参数 index 指定的索引位置的对象，索引下标从 0 开始。

Map 映射是一种把关键字对象映射的集合，他的每一个元素都包括一堆键对象和值对象，Map 没有继承 Collection 接口，从 Map 集合中检索元素时只要给出键对象，就会返回对应的值对象。

HashMap：Map 基于散列表的实现，插入和查询“键值对”的开销是固定的，可以通过构造器设置容量 capacity 和负载因子 load factor ，以调整容器的性能；

LinkedHashMap：类似于 HashMap，但在迭代遍历时，取得“键值对”的顺序是其插入次序，只比 HashMap 慢一点，而在迭代访问时反而更快，因为它使用链表维护内部次序。

TreeMap：基于红黑树数据结构的实现，查看“键”或“键值对”时，他们会对其排序（次序由 Comparabel 和 Comparator 决定）

15.ArrayMap 和 HashMap 的区别

ArrayMap 相比传统的 HashMap 速度更慢，因为其查找方法是二分法，并且当删除或添加数据时，会对空间重新调整，可以说 ArrayMap 是牺牲了时间来换空间，ArrayMap 与 HashMap 的区别主要在：

存储方式不同：HashMap 内部有一个 HashMapEntry<K,V>[ ]对象，而 ArrayMap 是一个<key,value>映射的数据结构，内部使用两个数组进行数据存储，一个数组记录 key 的 hash 值，另一个数组记录 value 值。

添加数据时扩容的处理不一样：HashMap 进行了 new 操作，重新创建对象，开销很大，而 ArrayMap 用的是 copy 数据，效率相对高很多。

ArrayMap 提供了数组收缩的功能，在 clear 或 remove 之后，会重新收缩数组，释放空间。

ArrayMap 采用的是二分法查找。

16.HashMap 和 HashTable 的区别

HashMap 是基于哈希表实现的，每一个元素是一个 key—value 对，其内部通过单链表解决冲突的问题 HashMap 是非线程安全的，只适用于单线程的环境下。多线程的环境下可以采用 concurrent 并发包下的 concurrentHashMap，HsahMap 实现了 serializable 接口，支持序列化，实现了 cloneable 接口，能被克隆。HashMap 内部维持了一个存储数据的 Entry 数组，HashMap 采用链表解决冲突，HashMap 中的 key 和 value 都允许为 null，key 为 null 的键值对永远都放在以 table[0]为节点的链表中。

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，容量不足时，同样会自动增大，HashTble 是线程安全的，能用在多线程的环境下，HashTable 实现了 serializable

接口，它支持序列化，实现了 cloneable 接口，能被克隆。

HashMap 和 HashTable 之间的区别有以下几点

继承的父类不同，hashTable 继承自 Dictionary 类，而 HashMap 继承自 AbstractMap 类，但二者都实现了 Map 接口。

线程安全性不同，HashTable 中的方法是 synchronized 的，而 HashMap 中的方法在缺省的情况下是非 ynchronized 的，在多线程的环境下，可以直接使用 HsahTable，不需要为他的方法实现同步，但使用 HashMap 时就必须自己增加同步处理。

key 和 value 是否允许为 null 值：关于其中的 key 和 value 都是对象，并且不能包含重复的 key，但可以包含重复的 value，hashtable 中，key 和 value 都不允许出现 null 值，但在 hashmap 中，null 可以作为键，这样的键只有一个，可以有多个键对应的值为 null.

17.HashMap 和 HashSet 的区别

HashMap：其实现了 Map 接口，HashMap 存储键值对，使用 put(　)方法将元素放入到 Map 中，HashMap 使用键对象来计算 hashcode 值，HashMap 比较快，因为是使用唯一的键来获取对象。

HashSet：实现了 Set 接口，hashSet 仅仅存储对象，使用 add（）方法将元素放入到 set 中，hashset 使用成员对象来计算 hashcode 值，对于两个对象来说，hashcode 可能相同，所以 equal 方法用来判断对象的相等性，如果两个对象不同的话，那么返回 false，hashSet 较 hashMap 来说较慢。

18.ArrayList 和 LinkedList 的区别

ArrayList 和 LinkedList，前者是 Array(动态数组)的数据结构，后者是 Link(链表)的数据结构，此外他们两个都是对 List 接口的实现

当随机访问 List 时（get 和 set 操作），ArrayList 和 LinkedList 的效率更高，因为 LinkedList 是线性的数据存储方式，所以需要移动指针从前往后查找。

当对数据进行增删的操作时（add 和 remove），LinkedList 比 ArrayList 的效率更高，因为 ArrayList 是数组，所以在其中进行增删操作时，会对操作点之后的所有数据的下标索引造成影响，需要进行数据的移动

从利用效率来看，ArrayList 自由性较低，因为需要手动的设置固定大小的容量，但是他的使用比较方便，只需要创建，然后添加数据，通过调用下标进行使用，而 LinkedList 自由性交给，能够动态的随数据量的变化而变化，但是它不便于使用。

19．数组和链表的区别

数组：是将元素在内存中连续的存储的，因为数据是连续存储的，内存地址连续，所以在查找数据的时候效率比较高，但在存储之前，需要申请一块连续的内存空间，并且在编译的时候就必须确定好他的空间大小。在运行的时候空间的大小是无法随着需要进行增加和减少的，当数据比较大时，有可能出现越界的情况，当数据比较小时，有可能浪费内存空间，在改变数据个数时，增加、插入、删除数据效率比较低。

链表：是动态申请内存空间的，不需要像数组需要提前申请好内存的大小，链表只需在使用的时候申请就可以了，根据需要动态的申请或删除内存空间，对于数据增加和删除以及插入比数组灵活，链表中数据在内存中可以在任意的位置，通过应用来关联数据。

20．Java 中多线程实现的三种方式

Java 中多线程实现的方式主要有三种：继承 Thread 类、实现 Runnable 接口、使用 ExecutorService、Callable、Future 实现有返回结果的多线程，其中前两种方式线程执行完没有返回值，只有最后一种是带返回值的。

继承 Thread 类实现多线程：继承 Thread 类本质上也是实现 Tunnable 接口

的一个实例，他代表一个线程的实例，并且启动线程的唯一方法是通过 Thread 类的 start（）方法，start（）方法是一个 native 方法，他将启动一个新线程，并执行 run( )方法。

实现 Runnable 接口方式实现多线程：实例化一个 Thread 对象，并传入实现的 Runnable 接口，当传入一个 Runnable target 参数给 Thread 后，Thraed 的 run（）方法就会调用 target.run( );

使用 ExecutorService、Callable、Future 实现有返回结果的多线程：可返回值的任务必须实现 Callable 接口，类似的无返回值的任务必须实现 Runnable 接口，执行 Callable 任务后，可以获取一个 Future 的对象，在该对象上调用 get 就可以获取到 Callable 任务返回的 Object 了，在结合线程池接口 ExecutorService 就可以实现有返回结果的多线程。

21.Java 中创建线程的三种方式

Java 中使用 Thread 类代表线程，所有的线程对象都必须时 Thread 类或其子类的实例，Java 中可以用三种方式来创建线程

继承 Java 中的 Thread 类创建线程：定义 Thread 类的子类，并重写其 run( )方法，run( )方法也称为线程执行体，创建该子类的实例，调用线程的 start（）方法启动线程。

实现 Runnable 接口创建线程：定义 Runnable 接口的实现类，重写 run（）方法，run 方法是线程的执行体，创建 Runnable 实现类的实例，并用这个实例作为 Thread 的 target 来创建 Thread 对象，这个 Thread 对象才是真正的线程对象，调用线程对象的 Start 方法启动线程。

使用 Callable 和 Future 创建线程：Callable 接口提供了一个 call( )方法，作为线程的执行体，call( )方法可以有返回值，call( )方法可以声明抛出异常，其创建线程并启动的步骤，创建 Callable 接口的实现类，并实现 call( )方法，创建该实现类的实例，使用 FutureTask 类来包装 Callable 对象，该 FuutureTask 对象封

装了 callable 对象的 call( )方法的返回值，使用 FutureTask 对象作为 Thread 对象的 target 创建并启动线程，调用 FutureTask 对象的 get( )方法来获得子线程执行结束后的返回值。

22.线程和进程的区别

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务，不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间，注意勿与栈内存混淆，每个线程都拥有单独的栈内存用来存储本地数据。

30.如何控制某个方法允许并发访问线程的个数

在 Java 中常使用 Semaphore（信号量）来进行并发编程，Semaphore 控制的是线程并发的数量，实例化一个 Semaphore 对象，如 Semaphore semaphore = newSemaphore(5,true) ,其创建了对象 semaphore，并初始化了 5 个信号量，即最多允许 5 个线程并发访问，在执行的任务中，调用 semaphore 的 acquire（）方法请求一个信号量，这时信号量个数就减 1，（一旦没有可使用的信号量，再次请求就会阻塞），来执行任务，执行完任务，调用 semaphore 的 release（）方法释放一个信号量此时信号量的个数就会加 1 。

23.什么是线程阻塞？线程该如何关闭？

阻塞式方法是指程序会一直等待该方法完成执行，而在此期间不做其他的事情，例如 ServerSocket 的 accept( )方法就是一直等待客户端连接，这里的阻塞是指调用结果返回之前，当前线程会被挂起，直到得到结果之后才返回你，此外还有异步和非阻塞式方法在任务完成前就返回。

线程关闭的方法有如下两种：

一种是调用线程的 stop（ ）方法；

另一种是自己自行设计一个停止线程的标记；

24.如何保证线程的安全

使用 Synchronized 关键字：

调用 Object 类的 wait 很 notify;

通过 ThreadLocal 机制实现；

25.实现线程同步的方式

Java 允许线程并发控制，当多个线程同时操作一个可共享的资源变量时（如数据的增删改查），将会导致数据的不准确，相互之间产生冲突，因此在加入同步锁以避免在该线程没有完成操作之前，被其他线程调用，从而保证该变量的唯一性和准确性，同步的方法有以下几种：

Synchronized 关键字修饰的方法：由于 Java 的每个对象都有一个内置锁，当用此关键字修饰方法时，内置锁会保护整个方法；

Synchronized 关键字修饰语句块：被 Synchronized 关键字修饰的语句块会自动加上内置锁，从而实现同步；

使用特殊域变量（volatile）实现线程同步：当一个共享变量被 volatile 修饰时，他会保证修改的值立即被更新到主存中，volatile 的特殊性在于，内存可见性，就是一个线程对于 volatile 变量的修改，对于其他线程来说是可见的，即线程每次获取 volatile 变量的值都是最新的。

26.Java 中锁与同步的相关知识

锁提供了两种主要的特性：互斥和可见性

互斥即一次只允许一个线程持有某个特定的锁，因此可使用该特性实现对共享数据的协调访问协议，这样，一次就只有一个线程能够使用该共享的数

据；可见性在必须确保锁释放之前对共享对象做出的更改对于随后获得该锁的另一个线程是可见的。

在 Java 中，为了确保多线程读写数据时的一致性，可以采用两种方式

同步：如使用 synchronized 关键字，或者使用锁对象；

使用 volatile 关键字：使变量的值发生改变时尽快通知其他线程；

Volatile 关键字详解

编译器为了加快程序的运行的速度，对一些变量的写操作会先在寄存器或者 CPU 缓存上进行，最后写入内存中，而在这个过程中，变量的新值对于其他线程是不可见的，当对使用 volatile 标记的变量进行修改时，会将其它缓存中存储的修改前的变量清除，然后重新读取。

27.Java 的原子性、可见性、有序性的理解

原子性：原子是世界上最小的物质单位，具有不可分割性，比如 a=0，这个操作是不可分割的，那么我们就会说这个操作是原子操作，再如 a++，这个操作实际上是 a=a+1，是可以分割的，所以他不是一个原子操作，非原子操作都会存在线程安全的问题，需要使用 synchronized 同步技术来使其变成一个原子操作，一个操作是原子操作，那么我么称它具有原子性。

可见性：是指线程之间的可见性，一个线程修改的状态对于另一个线程是可见的，比如用 volatile 修饰的变量就具有可见性，volatile 修饰的变量不允许线程内部缓存和重排序，即会直接修改内存，所以对其他线程是可见的，但 volatile 只能让其被修饰的内容具有可见性，并不能保证它具有原子性，因为 volatile 仅能使用在变量级别，并不能对方法进行修饰，

有序性：即线程执行的顺序按代码的先后顺序执行，在 Java 内存模型中，允许编译器和处理器对指令进行重排序，但重排序过程不会影响到单线程程序的执行，却会影响到多线程并发执行的正确性，在 Java 里面可以通过 volatile

关键字来保证一定的“有序性”，另外还可以通过 synchronized 和 Lock 来保证有序性。

28．Java 中死锁的概念，其产生的四个必要条件

死锁是一种情形，多个线程被阻塞，他们中的一个或者全部都在等待某个资源被释放，由于线程被无限期的阻塞，因此程序不能正常运行，简单的说就是线程死锁时，第一个线程等待第二个线程释放资源，而同时第二个线程又在等待第一个线程释放资源；

死锁产生的四个必要条件：

互斥条件：一个资源每次只能被一个进程使用，即在一段时间内某资源仅为一个进程所占有，此时若有其他请求该资源，则请求进程只能等待；

请求与保持条件：进程已经持有了至少一个资源，但又提出了新的资源请求，而该资源已经被其他进程所占有，此时请求会被阻塞，但对自己获得的资源又保持不放；

不可剥夺条件：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，只能由获得该资源的进程自己来释放（只能时主动释放）；

循环等待条件：即若干进程形成首尾相接的循环等待资源的关系，即形成了一个进程等待环路，环路中每一个进程所占有的资源同时被另一个进程所申请，也就是前一个进程占有后一个进程所申请的资源

这四个条件是死锁产生必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁；

死锁的避免：避免死锁的基本思想是系统对进程发出的每一个系统能够满足的资源申请进行动态检查，并根据检查结果决定是否分配资源，如果分配资源后系统可能发生死锁，则不分配，否则予以分配，这是一种保证系统不进入死锁状态的动态策略。

29.Java 中堆和栈的理解

在 Java 中内存分为两种，一种是栈内存，另一种是堆内存

堆内存：用于存储 Java 中的对象和数组，当我们 new 一个对象或创建一个数组的时候，就会在堆内存中开辟一段空间给它，用于存放，堆内存的特点：先进先出，后今后出，②可以动态的分配内存的大小，生存期不必告诉编译器，但存取速度较慢；

栈内存：主要用来执行程序用，比如基本类型的变量和对象的引用变量，其特点：①先进后出，后进后出，②存取速度比堆快，仅次于寄存器，栈数据可以共享，但其在栈中的数据大小和生存期必须是确定的；

栈内存和堆内存都属于 Java 内存的一种，系统会自动去回收它，但对于堆内存开发人员一般会自动回收。

栈是一块和线程紧密相关的内存区域，每个线程都有自己的栈内存，用于存储本地变量、方法参数和栈调用一个线程中存储的变量，对于其他线程是不可见的，而堆是所有线程共享的一个公用内存区域，对象都在堆里创建，但为了提升效率，线程会从堆中拷贝一个缓存到自己的栈中，如果多个线程使用该变量，就可能引发问题，这是 volatile 修饰变量就可以发挥作用，他要求线程从主存中读取变量的值。

30.理解线程间通信

线程是 CPU 调度的最小单位（进程是 CPU 分配资源的最小单位），在 Android 中主线程是不能够做耗时的操作的，子线程是不能更新 UI 的，而线程间的通信方式有很多，比如广播、接口回掉等，在 Android 中主要使用 handler，handler 通过调用 sendMessage 方法，将保存好的消息发送到 MessageQueue 中，而 Looper 对象不断地调用 loop 方法，从 MessageQueue 中取出 message，交给 handler 处理，从而完成线程间通信。

31.线程中的 join（）方法的理解，及如何让多个线程按顺序执行

Thread 类的 join（）方法主要作用是同步，它可以使线程之间的并行执行变为串行执行，join（）方法把指定的线程加入到当前线程中，可以将两个交替执行的线程合并为顺序执行的线程，比如在线程 B 中调用了线程 A 的 join（）方法，则会等到 A 线程执行完，才会继续执行线程 B，join 方法还可以指定时长，表示等待该线程执行指定时长后再执行其他线程，如 A.join(1000)，表示等待 A 线程执行 1000 毫秒后，再执行其他线程，当有多个线程、要使他们案顺序执行时，可以使用 join()方法在一个线程中启动另一个线程，另外一个线程在该线程执行完之后再继续执行。

32.工作者线程（workerThread）与主线程（UI 线程）的理解

Android 应用的主线程（UI 线程）肩负着绘制用户界面，和及时响应用户操作的重任，为避免“用户点击按钮后没有反应”的状况，就要确保主线程时刻保持着较高的响应性，把耗时的任务移除主线程，交予工作者线程（即子线程）完成，常见的工作者线程有 AsyncTask（异步任务）、IntentService、HandlerThread，他们本质上都是对线程或线程池的封装。

33.AsyncTask（异步任务）的工作原理

AsyncTask 是对 Handler 和线程池的封装，使用它可以更新用户界面，当然，这里的更新操作还是在主线程中完成的，但由于 AsyncTask 内部包含了一个 Handler，所以可以发送消息给主线程，让他更新 UI，另外 AsyncTask 内还包含了一个线程池，避免了不必要的创建和销毁线程的开销。

34.并发和并行的区别及理解

在单核的机器上，“多进程”并不是真正多个进程同时执行，而是通过 CPU 时间分片，操作系统快速在进程间切换而模拟出来的多进程，我们通常将这种情况称为并发，也就是多个进程的运行行为是“一并发生”的，但不是同时执行的，因为 CPU 核数的限制（CPU 和通用寄存器只有有一套），严格来说，同一时刻只能存在一个进程的上下文。

但在多核 CPU 上，能真正实现多个进程并行执行，这种情况叫做并行，因为多个进程是真正“一并执行的”(具体多少个进程可以并行执行取决于 CPU 的核数)，所以可知，并发是一个比并行更加宽泛的概念，在单核情况下，并发只是并发，而在多核情况下，并发就变为并行了。

35.同步和异步的区别、阻塞和非阻塞的区别的理解

同步：所谓同步是一个服务的完成需要依赖其他服务，只有等待被依赖的服务完成后，依赖的服务才能算完成，这是一种可靠的服务序列，要么都成功，要么都失败服务的状态可以保持一致；

异步：所谓异步是一个服务的完成需要依赖其他的服务时，只要通知其依赖的服务开始执行，而不需要等待被依赖的服务完成，此时该服务就算完成了，至于被依赖的服务是否真正完成并不关心，所以这是不可靠的服务序列；

阻塞：阻塞调用是指调用结果返回之前，当前线程会被挂起，一直处于等待消息通知，不能够执行其他业务，函数只有在得到结果之后才会返回；

非阻塞：和阻塞的概念相对应，指在不能立刻得到结果之前，该函数不会阻塞当前线程，而会立即返回；

阻塞调用和同步调用是不同的，对于同步调用来说，很多时候当前线程可能还是激活的，只是从逻辑上当前函数没有返回值而已，此时这个线程可能也会处理其他消息，所以如下总结：

如果这个线程在等待当前函数返回时，仍在执行其他消息处理，这种情况叫做同步非阻塞；

如果这个线程在等待当前函数返回时，没有执行其他的消息处理，而是处于挂起等待的状态，这种情况叫做同步阻塞；

如果这个线程当前的函数已经返回，并且仍在执行其他的消息处理，这种情况叫做异步非阻塞；

如果这个线程当前的函数已经返回，但没有执行其他的消息处理，而是处于被挂起的等待状态，这种情况叫做异步阻塞；

同步与异步的重点在于的等待依赖的服务是否返回结果（即使没有执行完），也就是结果通知的方式，而不管其依赖的服务是否完成，阻塞与非阻塞的重点在于当前线程等待消息返回时的行为，是否执行其他的消息处理，当前线程是否被挂起；

36.Java 中任务调度的理解

大部分操作系统（如 windows、Linux）的任务调度采用时间片轮转的抢占式调度方式，也就是说一个任务执行一小段时间后强制暂停去执行下一个任务，每个任务轮流执行，任务执行的一段时间叫做时间片，任务正在执行的状态叫做运行状态，任务执行一段时间后强制暂停去执行下一个任务，被暂停的任务就处于就绪状态，等待下一个属于他的时间片的到来，这样每个任务都可以得到执行，由于 CPU 的执行效率非常高，时间片非常短，在各个任务之间快速的切换，给人的感觉是多个任务在“同时执行”，这也即我们所说的并发；

37.Java 中进程的详细概念

计算机的额核心是 CPU，它承担了所有的计算任务，操作系统是计算机的管理者，它负责任务的调度、资源的分配和管理，统领整个计算机的硬件，应用程序是具有某种功能的程序，其运行在操作系统上，而进程则是一个具有一定独立功能的程序在一个数据集上一次动态执行的过程，是操作系统进行资源分配和调度的一个独立单位，是应用程序运行的载体，进程是 CPU 资源分配的

最小单位（线程是 CPU 执行的最小的单元），各个进程之间内存地址相互隔离。