# Java基础点

### 0，面对对象OOP和面对过程的区别？

面对过程 \* 优点：性能比面对对象高，类调用需要实例化导致开销大，耗资源；比如单片机，嵌入式，Linux、Unix一般采用面对过程，他们对性能要求极高 \* 缺点：维护困难，不容易复用，不容易扩展

面对对象，优点就是面对过程的缺点；易维护，复用，扩展；缺点就是面对过程的优点；性能比面对过程低；

面对对象的特征：封装，继承和多态 - 封装：把对象的属性私有化，同时提供一些可以被外界访问该属性的方法 - 继承：是使用已存在的类的定义作为基础建立新类的技术，复用已存在的代码 - 多态：程序定义的引用变量所指向的具体类型不确定，在运行时才确定；它的2种形式：继承和接口

### 1，java的8种基本数据类型，以及int，char，long 各占多少字节数？

char 2个字节，int 4个字节，long 8个字节。



image.png

### 2、int与integer的区别

int 基本类型

integer 对象 int的封装类

### 3，String、StringBuffer、StringBuilder区别

String:字符串常量 不适用于经常要改变值得情况，每次改变相当于生成一个新的对象；用final关键字字符数组保存字符串，导致它是不可更改，不可继承的常量，private final char value[]

StringBuffer:字符串变量 （线程安全） 对方法加了同步锁所以是安全

StringBuilder:字符串变量（线程不安全） 确保单线程下可用，效率略高于StringBuffer，对方法没有加同步锁

StringBuffer和StringBuilder都是继承自AbstractStringBuilder类，此类也是使用字符数组保存字符串char[] value，但是没有用final关键字修饰，所以这2种对象都是可变的。

对于三者使用的总结：

1. 操作少量的数据 = String

2. 单线程操作字符串缓冲区下操作大量数据 = StringBuilder

3. 多线程操作字符串缓冲区下操作大量数据 = StringBuffer

扩展：String为什么要设计成不可变的？

1、字符串池的需求

字符串池是方法区（Method Area）中的一块特殊的存储区域。当一个字符串已经被创建并且该字符串在 池 中，该字符串的引用会立即返回给变量，而不是重新创建一个字符串再将引用返回给变量。如果字符串不是不可变的，那么改变一个引用（如: string2）的字符串将会导致另一个引用（如: string1）出现脏数据。

2、允许字符串缓存哈希码

在java中常常会用到字符串的哈希码，例如： HashMap 。String的不变性保证哈希码始终一，因此，他可以不用担心变化的出现。 这种方法意味着不必每次使用时都重新计算一次哈希码——这样，效率会高很多。

3、安全

String广泛的用于java 类中的参数，如：网络连接（Network connetion），打开文件（opening files ）等等。如果String不是不可变的，网络连接、文件将会被改变——这将会导致一系列的安全威胁。操作的方法本以为连接上了一台机器，但实际上却不是。由于反射中的参数都是字符串，同样，也会引起一系列的安全问题。

### 4，什么是内部类？内部类的作用

内部类：在一个类的内部定义另外一个类，被嵌套的类就是内部类。

内部类可直接访问外部类的属性

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内)、匿名内部类（没构造方法）、静态内部类（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）

1，内部类可以用多个实例，每个实例都有自己的状态信息，并且与其他外围对象的信息相互独立

2，在单个外围类中，可以让多个内部类以不同的方式实现同一个接口，或者继承同一个类。

3，创建内部类对象的时刻并不依赖于外围类对象的创建

4，内部类提供更好的封装，除了该外围类，其他类都不能访问

内部类在编译完后也会产生.class文件，但文件名称是：外部类名称$内部类名称.class。

### 5，final，finally，finalize的区别

* final:修饰类、成员变量和成员方法，类不可被继承，成员变量不可变，成员方法不可重写
* finally:与try...catch...共同使用，确保无论是否出现异常都能被调用到
* finalize:类的方法,垃圾回收之前会调用此方法,子类可以重写finalize()方法实现对资源的回收

扩展：static

1、static变量：对于静态变量在内存中只有一个拷贝（节省内存），JVM只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问（方便），当然也可以通过对象来访问（但是这是不推荐的）。

2、static代码块： static代码块是类加载时，初始化自动执行的。

3、static方法：static方法可以直接通过类名调用，任何的实例也都可以调用，因此static方法中不能用this和super关键字，不能直接访问所属类的实例变量和实例方法(就是不带static的成员变量和成员成员方法)，只能访问所属类的静态成员变量和成员方法。

### 6，java中==和equals和hashCode的区别

基本数据类型的== 是比较的值是否相等.

类的==比较的内存的地址，即是否是同一个对象，在不覆盖equals的情况下，同比较内存地址，原实现也为 == ，如String等重写了equals方法.

hashCode也是Object类的一个方法。返回一个离散的int型整数。在集合类操作中使用，为了提高查询速度。（HashMap，HashSet等比较是否为同一个）

如果两个对象equals，Java运行时环境会认为他们的hashcode一定相等。 如果两个对象不equals，他们的hashcode有可能相等。 如果两个对象hashcode相等，他们不一定equals。 如果两个对象hashcode不相等，他们一定不equals。

eg：

public class test1 {
 public static void main(String[] args) {
 String a = new String("ab"); // a 为一个引用 String b = new String("ab");
 // b 为另一个引用,对象的内容一样
 String aa = "ab"; // 放在常量池中 String bb = "ab";
 // 从常量池中查找
 if(aa==bb)//true
 System.out.println("aa==bb")
 if(a == b)//false,非同一对象
 System.out.println("a==b")
 if(a.equals(b))//true
 System.out.println("aEqb")
 if(42 == 42.0)
 System.out.println("true")
 }
}

### 7，哪些情况下的对象会被垃圾回收机制处理掉

1. 所有实例都没有活动线程访问。
2. 没有被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。

要判断怎样的对象是没用的对象。这里有2种方法：

* 采用标记计数的方法：

给内存中的对象给打上标记，对象被引用一次，计数就加1，引用被释放了，计数就减一，当这个计数为0的时候，这个对象就可以被回收了。当然，这也就引发了一个问题：循环引用的对象是无法被识别出来并且被回收的。所以就有了第二种方法：

* 采用根搜索算法：从一个根出发，搜索所有的可达对象，这样剩下的那些对象就是需要被回收的

Java垃圾回收机制最基本的做法是分代回收。

内存中的区域被划分成不同的世代，对象根据其存活的时间被保存在对应世代的区域中。一般的实现是划分成3个世代：年轻、年老和永久。内存的分配是发生在年轻世代中的。当一个对象存活时间足够长的时候，它就会被复制到年老世代中。对于不同的世代可以使用不同的垃圾回收算法。进行世代划分的出发点是对应用中对象存活时间进行研究之后得出的统计规律。一般来说，一个应用中的大部分对象的存活时间都很短。比如局部变量的存活时间就只在方法的执行过程中。基于这一点，对于年轻世代的垃圾回收算法就可以很有针对性。

### 8，父类的静态方法能否被子类重写是否可以被继承，为什么？

可以继承，但是不能重写，

如果子类里面定义了静态方法和属性，那么这时候父类的静态方法或属性称之为"隐藏"。

如果你想要调用父类的静态方法和属性，直接通过父类名.方法或变量名完成。

构造器Constructor是否可被override，不能，但是可以overload重载

### 9，抽象类的意义

便于维护，只有子类继承抽象类的所有方法 把相同的属性和方法抽象，便于代码的维护

为子类提供一个公共类型，封装子类中的重复内容定义抽象方法，子类虽然有不同的实现，但是定义是一致的

扩展：接口和抽象类的区别？

* 抽象类可以提供成员方法实现的细节，而接口只能存在抽象方法。
* 抽象类的成员变量可以是各种类型，而接口中的成员变量只能是public static finla类型。
* 接口中不能含有静态方法及静态代码块，而抽象类可以有静态方法和静态代码块。
* 抽象类可以有非抽象方法
* 一个类可以实现多个接口，但最多只能实现一个抽象类
* 一个如果实现接口就得实现接口所有的方法，而抽象类不用……
* 接口中的实例变量默认是final类型的，而抽象类中则不一定

### 10，接口的意义和特点

* 增加扩展性，abstract class 和interface 是支持抽象类定义的两种机制
* 便于维护
* 规范
* 接口是实现软件松耦合
* 弥补了java无法多继承的缺陷
* 安全、严密性（继承接口不需要知道内部怎么操作的）

### 11，什么是多态，如何实现多态，Java中实现多态的机制是什么？

事物在运行过程中存在不同的状态叫多态

多态：父类声明指向子类对象，即引用变量在程序编写的时候不确定，在程序运行的时候才能确定。

实现：子类继承父类并且覆写父类中的方法，或者说实现接口 作用：消除类型之间的耦合关系。实现多态的必要条件：继承、重写

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性的一种表现 重载Overloading是一个类中多态性的一种表现

参考：<https://www.runoob.com/java/java-override-overload.html>



image.png

### 12，Serializable 和Parcelable 的区别

* Serializable Java 序列化接口 在硬盘上读写 读写过程中有大量临时变量的生成，引发频繁GC，其本质上使用了反射，序列化过程慢，内部执行大量的i/o操作，效率很低。
* Parcelable Android 序列化接口 效率高 使用麻烦 在内存中读写（AS有相关插件 一键生成所需方法） ，对象不能保存到磁盘中

选择原则：

若仅在内存中使用，如activity，优先使用Parcelable，它性能高。若是持久化操作，优先使用Serializable

扩展：

序列化：将一个对象转换成可存储或可传输的状态，序列化后的对象可以在网络上传输，也可以存储到本地，或实现跨进程传输。

为什么要序列化：开发过程中，我们要将对象的引用传给其他Activity或Fragment使用时，需要将对象放到一个Intent或Bundle中，再进行传递，而Intent或Bundle只能识别基本数据类型和被序列化的类型。

### 13，List,Set,Map的区别

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接口主要实现了两个实现类：

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

List的特征是其元素以线性方式存储，集合中可以存放重复对象。

* ArrayList() : 代表长度可以改变得数组。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(): 在实现中采用链表数据结构。插入和删除速度快，访问速度慢。

Map 是一种把键对象和值对象映射的集合，它的每一个元素都包含一对键对象和值对象。

Map没有继承于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，但是迭代遍历它时，取得“键值对”的顺序是其插入次序，或者是最近最少使用(LRU)的次序。只比HashMap慢一点。而在迭代访问时发而更快，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，它们会被排序(次序由Comparabel或Comparator决定)。TreeMap的特点在 于，你得到的结果是经过排序的。TreeMap是唯一的带有subMap()方法的Map，它可以返回一个子树。
* WeakHashMap ：弱键(weak key)Map，Map中使用的对象也被允许释放: 这是为解决特殊问题设计的。如果没有map之外的引用指向某个“键”，则此“键”可以被垃圾收集器回收。

### 14、ArrayMap和HashMap的对比

1、存储方式不同，HashMap内部有一个HashMapEntry[]对象，每一个键值对都存储在这个对象里，当使用put方法添加键值对时，就会new一个 HashMapEntry对象，

2、添加数据时扩容时的处理不一样，进行了new操作，重新创建对象，开销很大。ArrayMap用的是copy数据，所以效率相对要高。

3、ArrayMap提供了数组收缩的功能，在clear或remove后，会重新收缩数组，是否空间

4、ArrayMap采用二分法查找；

### 15、HashMap和HashTable的区别

HashMap不是线程安全的，效率高一点、方法不是Synchronize的要提供外同步，有containsvalue和containsKey方法。

Hashtable是，线程安全，不允许有null的键和值，效率稍低，方法是是Synchronize的。有contains方法方法。Hashtable 继承于Dictionary 类

### 16、HashMap与HashSet的区别

* HashMap实现了Map接口,HashMap储存键值对,使用put()方法将元素放入map中,HashMap中使用键对象来计算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equals()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来说比较慢。

### 17、HashSet与HashMap怎么判断集合元素重复？

HashSet不能添加重复的元素，当调用add（Object）方法时候，首先会调用Object的hashCode方法判hashCode是否已经存在，如不存在则直接插入元素； 如果已存在则调用Object对象的equals方法判断是否返回true，如果为true则说明元素已经存在，如为false则插入元素。

### 18、ArrayList和LinkedList的区别，以及应用场景

ArrayList是基于数组实现的，ArrayList线程不安全。LinkedList是基于双链表实现的：

使用场景：

* 如果应用程序对各个索引位置的元素进行大量的存取或删除操作，ArrayList对象要远优于LinkedList对象；
* 如果应用程序主要是对列表进行循环，并且循环时候进行插入或者删除操作，LinkedList对象要远优于ArrayList对象；

读写时间的复杂度

（1）ArrayList：ArrayList是一个泛型类，底层采用数组结构保存对象。数组结构的优点是便于对集合进行快速的随机访问，即如果需要经常根据索引位置访问集合中的对象，使用由ArrayList类实现的List集合的效率较好。数组结构的缺点是向指定索引位置插入对象和删除指定索引位置对象的速度较慢，并且插入或删除对象的索引位置越小效率越低，原因是当向指定的索引位置插入对象时，会同时将指定索引位置及之后的所有对象相应的向后移动一位。

（2）LinkedList：LinkedList是一个泛型类，底层是一个双向链表，所以它在执行插入和删除操作时比ArrayList更加的高效，但也因为链表的数据结构，所以在随机访问方面要比ArrayList差。

ArrayList 是线性表（数组）

get() 直接读取第几个下标，复杂度 O(1)

add(E) 添加元素，直接在后面添加，复杂度O（1）

add(index, E) 添加元素，在第几个元素后面插入，后面的元素需要向后移动，复杂度O（n）

remove（）删除元素，后面的元素需要逐个移动，复杂度O（n）

LinkedList 是链表的操作

get() 获取第几个元素，依次遍历，复杂度O(n)

add(E) 添加到末尾，复杂度O(1)

add(index, E) 添加第几个元素后，需要先查找到第几个元素，直接指针指向操作，复杂度O(n)

remove（）删除元素，直接指针指向操作，复杂度O(1)

### 19、数组和链表的区别

* 数组：是将元素在内存中连续存储的；
* 优点：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；
* 缺点：在存储之前，我们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它的空间的大小。在运行的时候空间的大小是无法随着你的需要进行增加和减少而改变的，当数据两比较大的时候，有可能会出现越界的情况，数据比较小的时候，又有可能会浪费掉内存空间。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，不需要像数组需要提前申请好内存的大小，链表只需在用的时候申请就可以，根据需要来动态申请或者删除内存空间，对于数据增加和删除以及插入比数组灵活。还有就是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（就是通过存在元素的指针来联系）

### 20，HashMap如何保证元素均匀分布

hash & (length-1)

通过Key值的hashCode值和hashMap长度-1做与运算

hashmap中的元素，默认情况下，数组大小为16，也就是2的4次方，如果要自定义HashMap初始化数组长度，也要设置为2的n次方大小，因为这样效率最高。因为当数组长度为2的n次幂的时候，不同的key算出的index相同的几率较小，那么数据在数组上分布就比较均匀，也就是说碰撞的几率小，相对的，查询的时候就不用遍历某个位置上的链表，这样查询效率也就较高了

### 21，为什么HashMap线程不安全（hash碰撞与扩容导致）

HashMap的底层存储结构是一个Entry数组，每个Entry又是一个单链表，一旦发生Hash冲突的的时候，HashMap采用拉链法解决碰撞冲突，因为hashMap的put方法不是同步的，所以他的扩容方法也不是同步的，在扩容过程中，会新生成一个新的容量的数组，然后对原数组的所有键值对重新进行计算和写入新的数组，之后指向新生成的数组。当多个线程同时检测到hashmap需要扩容的时候就会同时调用resize操作，各自生成新的数组并rehash后赋给该map底层的数组table，结果最终只有最后一个线程生成的新数组被赋给table变量，其他线程的均会丢失。而且当某些线程已经完成赋值而其他线程刚开始的时候，就会用已经被赋值的table作为原始数组，这样也会有问题。扩容的时候 可能会引发链表形成环状结构

### 21. 自动装箱与拆箱

装箱：将基本类型用它们对应的引用类型包装起来；

拆箱：将包装类型转换为基本数据类型；

### 22，在一个静态方法内调用一个非静态成员为什么是非法的？

静态方法可以不通过对象进行调用，因此在静态方法里，不能调用其他非静态变量，也不可以访问非静态变量成员。

### 23，构造方法有那些特性？

* 名字和类名相同
* 没有返回值，不能用void声明构造函数
* 生成类的对象时自动执行，无需调用

### 24，泛型中extends和super的区别

<? extends T>限定参数类型的上界，参数类型必须是T或T的子类型，但对于List<? extends T>，不能通过add()来加入元素，因为不知道<? extends T>是T的哪一种子类；
<? super T>限定参数类型的下界，参数类型必须是T或T的父类型，不能能过get()获取元素，因为不知道哪个超类；

### 25，说下泛型原理，并举例说明

泛型就是将类型变成参数传入，使得可以使用的类型多样化，从而实现解耦。

Java泛型是在Java1.5以后出现的，为保持对以前版本的兼容，使用了擦除的方法实现泛型。

擦除是指在一定程度无视类型参数T，直接从T所在的类开始向上T的父类去擦除，如调用泛型方法，传入类型参数T进入方法内部，若没在声明时做类似 public T methodName(T extends Father t){}， Java就进行了向上类型的擦除，直接把参数t当做Object类来处理，而不是传进去的T。即在有泛型的任何类和方法内部，它都无法知道自己的泛型参数，擦除和转型都是在边界上发生，即传进去的参在进入类或方法时被擦除掉，但传出来的时候又被转成了我们设置的T。在泛型类或方法内，任何涉及到具体类型（即擦除后的类型的子类）操作都不能进行，如new T()，或者T.play()（play为某子类的方法而不是擦除后的类的方法）

### 26，说下你对Collection这个类的理解。

Collection是集合框架的顶层接口，是存储对象的容器,Colloction定义了接口的公用方法如add remove clear等等，它的子接口有两个，List和Set,List的特点有元素有序，元素可以重复，元素都有索引（角标），典型的有

Vector:内部是数组数据结构，是同步的（线程安全的）。增删查询都很慢。

ArrayList:内部是数组数据结构，是不同步的（线程不安全的）。替代了Vector。查询速度快，增删比较慢。

LinkedList:内部是链表数据结构，是不同步的（线程不安全的）。增删元素速度快。

而Set的是特点元素无序，元素不可以重复

HashSet：内部数据结构是哈希表，是不同步的。

Set集合中元素都必须是唯一的，HashSet作为其子类也需保证元素的唯一性。

判断元素唯一性的方式：

通过存储对象（元素）的hashCode和equals方法来完成对象唯一性的。

如果对象的hashCode值不同，那么不用调用equals方法就会将对象直接存储到集合中；

如果对象的hashCode值相同，那么需调用equals方法判断返回值是否为true，

若为false, 则视为不同元素，就会直接存储；

若为true， 则视为相同元素，不会存储。

如果要使用HashSet集合存储元素，该元素的类必须覆盖hashCode方法和equals方法。一般情况下，如果定义的类会产生很多对象，通常都需要覆盖equals，hashCode方法。建立对象判断是否相同的依据。

TreeSet：保证元素唯一性的同时可以对内部元素进行排序，是不同步的。

判断元素唯一性的方式：

根据比较方法的返回结果是否为0，如果为0视为相同元素，不存；如果非0视为不同元素，则存。

TreeSet对元素的排序有两种方式：

方式一：使元素（对象）对应的类实现Comparable接口，覆盖compareTo方法。这样元素自身具有比较功能。

方式二：使TreeSet集合自身具有比较功能，定义一个比较器Comparator，将该类对象作为参数传递给TreeSet集合的构造函数

# Java高级

### 1，静态代理和动态代理的区别，什么场景使用？

代理是一种常用的设计模式，目的是：为其他对象提供一个代理以控制对某个对象的访问，将两个类的关系解耦。代理类和委托类都要实现相同的接口，因为代理真正调用的是委托类的方法。

* 静态代理类：由程序员创建或由特定工具自动生成源代码，再对其编译。在程序运行前，代理类的.class文件就已经存在了。静态代理通常只代理一个类
* 动态代理类：在程序运行时，运用反射机制动态创建而成。动态代理代理的是一个接口下的多个实现类；
* 实现步骤：
* a.实现InvocationHandler接口创建自己的调用处理器；
* b.给Proxy类提供ClassLoader和代理接口类型数组创建动态代理类；
* c.利用反射机制得到动态代理类的构造函数；
* d.利用动态代理类的构造函数创建动态代理类对象；
* 使用场景：Retrofit中直接调用接口的方法；Spring的AOP机制；

### 2，说说你对Java反射的理解

JAVA反射机制是在运行状态中, 对于任意一个类, 都能够知道这个类的所有属性和方法; 对于任意一个对象, 都能够调用它的任意一个方法和属性。 这种能动态获取信息及动态调用对象方法的功能称为java语言的反射机制。

反射的作用：开发过程中，经常会遇到某个类的成员变量，方法或属性是私有的，或只对系统应用开放，这里就可以利用Java的反射机制通过反射来获取所需要的私有成员或者方法。

1. 获取类的Class对象实例 Class clz = Class.forName("com.zhenai.api.Apple");
2. 根据Class对象实例获取Constructor对象  Constructor appConstructor = clz.getConstructor();
3. 使用Constructor对象的newInstance方法获取反射类对象 Object appleObj = appConstructor.newInstance();
4. 获取方法的Method对象  Method setPriceMethod = clz.getMethod("setPrice", int.class);
5. 利用invoke方法调用方法  setPriceMethod.invoke(appleObj, 14);
6. 通过getFields()可以获取Class类的属性，但无法获取私有属性，而getDeclaredFields()可以获取到包括私有属性在内的所有属性。带有Declared修饰的方法可以反射到私有的方法，没有Declared修饰的只能用来反射公有的方法，其他如Annotation。

从对象出发，通过反射（Class类）可以取得类的完整信息（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自己的总结： 在运行过程中获得类、对象、方法的所有信息。

### 3，说说你对Java注解的理解

注解是通过@interface关键字来进行定义的，形式和接口差不多，只是前面多了一个@

public @interface TestAnnotation {

}

注解的作用：

1）提供信息给编译器：编译器可利用注解来探测错误和警告信息

2）编译阶段：软件工具可以利用注解信息来生成代码、html文档或做其它相应处理；

3）运行阶段：程序运行时可利用注解提取代码

注解是通过反射获取的，可以通过Class对象的isAnnotationPresent()方法判断它是否应用了某个注解，再通过getAnnotation()方法获取Annotation对象

使用时@TestAnnotation来引用，要使注解能正常工作，还需要使用元注解，它是可以注解到注解上的注解。元标签有@Retention @Documented @Target @Inherited @Repeatable五种

@Retention说明注解的存活时间，取值有RetentionPolicy.SOURCE 注解只在源码阶段保留，在编译器进行编译时被丢弃；RetentionPolicy.CLASS 注解只保留到编译进行的时候，并不会被加载到JVM中。RetentionPolicy.RUNTIME可以留到程序运行的时候，它会被加载进入到JVM中，所以在程序运行时可以获取到它们。

@Documented 注解中的元素包含到javadoc中去

@Target  限定注解的应用场景，ElementType.FIELD给属性进行注解；ElementType.LOCAL\_VARIABLE可以给局部变量进行注解；ElementType.METHOD可以给方法进行注解；ElementType.PACKAGE可以给一个包进行注解 ElementType.TYPE可以给一个类型进行注解，如类、接口、枚举

@Inherited 若一个超类被@Inherited注解过的注解进行注解，它的子类没有被任何注解应用的话，该子类就可继承超类的注解；

### 4，Java中堆和栈

* Java程序运行时的内存分配策略

静态存储区：主要存放静态数据，全局static数据和常量

栈区：方法体内的局部变量都在栈上创建

堆区：通常就是指在程序运行时直接new出来的内存

* 栈内存，堆内存的区别

在方法体内定义的（局部变量）一些基本类型的变量和对象的引用变量都是在方法的栈内存中分配的。

堆内存用来存放所有由new创建的对象（包括该对象其中的所有成员变量）和数组，在堆中分配的内存将由Java垃圾回收器来自动管理。

* Java内存回收机制



image.png

* Java内存泄露的原因

长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露

为什么把这个问题归类在多线程和并发面试题里？

因为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其它线程是不可见的。而堆是所有线程共享的一片公用内存区域。

对象都在堆里创建，为了提升效率线程会从堆中弄一个缓存到自己的栈，

如果多个线程使用该变量就可能引发问题，这时volatile 变量就可以发挥作用了，它要求线程从主存中读取变量的值。

### 5，进程和线程的区别

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。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。

进程是cpu资源分配的最小单位，线程是cpu调度的最小单位、是进程中运行的多个子任务。

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它资源。

一个进程内可拥有多个线程，进程可开启进程，也可开启线程。

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。

扩展1：开启线程的三种方式？

分别是继承Thread类重写run方法、实现Runable接口和使用线程池

扩展2：如何保证线程安全？

synchronized加锁关键字；Object方法中的wait,notify；ThreadLocal机制 来实现的。

扩展3：如何实现线程同步？

* 1、synchronized关键字修改的方法。
* 2、synchronized关键字修饰的语句块
* 3、使用特殊域变量（volatile）实现线程同步

扩展4：run()和start()方法区别

start()方法被用来启动新创建的线程，而且start()内部调用了run()方法，这和直接调用run()方法的效果不一样。

当你调用run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。

扩展5：如何控制某个方法允许并发访问线程的个数？

semaphore.acquire() 请求一个信号量，这时候的信号量个数-1（一旦没有可使用的信号量，也即信号量个数变为负数时，再次请求的时候就会阻塞，直到其他线程释放了信号量）

semaphore.release() 释放一个信号量，此时信号量个数+1

扩展6：什么导致线程阻塞？线程如何关闭？

阻塞式方法是指程序会一直等待该方法完成期间不做其他事情，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会返回。此外，还有异步和非阻塞式方法在任务完成前就返回。

如何关闭：一种是调用它里面的stop()方法；另一种就是你自己设置一个停止线程的标记 （推荐这种）；

使用中断interrupt(),但是它只是传递中断请求消息，并不代表立马停止目标线程。

扩展7：线程的状态？

New:新建状态，new出来，还没有调用start

Runnable:可运行状态，调用start进入可运行状态，可能运行也可能没有运行，取决于操作系统的调度

Blocked:阻塞状态，被锁阻塞，暂时不活动，阻塞状态是线程阻塞在进入synchronized关键字修饰的方法或代码块(获取锁)时的状态。

Waiting：等待状态，不活动，不运行任何代码，等待线程调度器调度，wait sleep

Timed Waiting:超时等待，在指定时间自行返回

Terminated:终止状态，包括正常终止和异常终止

扩展8：线程中sleep和wait的区别

(1)这两个方法来自不同的类，sleep是来自Thread，wait是来自Object；

(2)sleep方法没有释放锁，仅仅释放了CPU资源或者让当前线程停止执行一段时间，

而wait方法释放了锁，需要调用notify后才能继续进入锁。

(3)wait,notify,notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用。

### 6，线程间通信

我们知道线程是CPU调度的最小单位。在Android中主线程是不能够做耗时操作的，子线程是不能够更新UI的。而线程间通信的方式有很多，比如广播，Eventbus，接口回掉，在Android中主要是使用handler。handler通过调用sendmessage方法，将保存消息的Message发送到Messagequeue中，而looper对象不断的调用loop方法，从messageueue中取出message，交给handler处理，从而完成线程间通信。

### 7，Thread为什么不能用stop放停止

从SUN的官方文档可以得知，调用Thread.stop()方法是不安全的，这是因为当调用Thread.stop()方法时，会发生下面两件事：

1. 即刻抛出ThreadDeath异常，在线程的run()方法内，任何一点都有可能抛出ThreadDeath Error，包括在catch或finally语句中。

2. 释放该线程所持有的所有的锁。调用thread.stop()后导致了该线程所持有的所有锁的突然释放，那么被保护数据就有可能呈现不一致性，其他线程在使用这些被破坏的数据时，有可能导致一些很奇怪的应用程序错误。

### 8，线程同步机制原理

为什么需要线程同步？当多个线程操作同一个变量的时候，存在这个变量何时对另一个线程可见的问题，也就是可见性。每一个线程都持有主存中变量的一个副本，当他更新这个变量时，首先更新的是自己线程中副本的变量值，然后会将这个值更新到主存中，但是是否立即更新以及更新到主存的时机是不确定的，这就导致当另一个线程操作这个变量的时候，他从主存中读取的这个变量还是旧的值，导致两个线程不同步的问题。

线程同步就是为了保证多线程操作的可见性和原子性，比如我们用synchronized关键字包裹一端代码，我们希望这段代码执行完成后，对另一个线程立即可见，另一个线程再次操作的时候得到的是上一个线程更新之后的内容，还有就是保证这段代码的原子性，这段代码可能涉及到了好几部操作，我们希望这好几步的操作一次完成不会被中间打断，锁的同步机制就可以实现这一点。一般说的synchronized用来做多线程同步功能，其实synchronized只是提供多线程互斥，而对象的wait()和notify()方法才提供线程的同步功能。

JVM通过Monitor对象实现线程同步，当多个线程同时请求synchronized方法或块时，monitor会设置几个虚拟逻辑数据结构来管理这些多线程。新请求的线程会首先被加入到线程排队队列中，线程阻塞，当某个拥有锁的线程unlock之后，则排队队列里的线程竞争上岗（synchronized是不公平竞争锁，下面还会讲到）。如果运行的线程调用对象的wait()后就释放锁并进入wait线程集合那边，当调用对象的notify()或notifyall()后，wait线程就到排队那边。这是大致的逻辑。

### 9，线程池的种类

1.FixedThreadPool:可重用固定线程数的线程池，只有核心线程，没有非核心线程，核心线程不会被回收，有任务时，有空闲的核心线程就用核心线程执行，没有则加入队列排队

2.SingleThreadExecutor:单线程线程池，只有一个核心线程，没有非核心线程，当任务到达时，如果没有运行线程，则创建一个线程执行，如果正在运行则加入队列等待，可以保证所有任务在一个线程中按照顺序执行，和FixedThreadPool的区别只有数量

3.CachedThreadPool:按需创建的线程池，没有核心线程，非核心线程有Integer.MAX\_VALUE个，每次提交

任务如果有空闲线程则由空闲线程执行，没有空闲线程则创建新的线程执行，适用于大量的需要立即处理的并且耗时较短的任务

4.ScheduledThreadPoolExecutor:继承自ThreadPoolExecutor,用于延时执行任务或定期执行任务，核心线程数固定，线程总数为Integer.MAX\_VALUE

线程池ThreadPoolExecutor

线程池的工作原理：线程池可以减少创建和销毁线程的次数，从而减少系统资源的消耗，当一个任务提交到线程池时

1. 首先判断核心线程池中的线程是否已经满了，如果没满，则创建一个核心线程执行任务，否则进入下一步
2. 判断工作队列是否已满，没有满则加入工作队列，否则执行下一步
3. 判断线程数是否达到了最大值，如果不是，则创建非核心线程执行任务，否则执行饱和策略，默认抛出异常

### 10，原子性 可见性 有序性

原子性：对基本数据类型的读取和赋值操作是原子性操作，这些操作不可被中断，是一步到位的，例如x=3是原子性操作，而y = x就不是，它包含两步：第一读取x，第二将x写入工作内存；x++也不是原子性操作，它包含三部，第一，读取x，第二，对x加1，第三，写入内存。原子性操作的类如：AtomicInteger AtomicBoolean AtomicLong AtomicReference

可见性：指线程之间的可见性，既一个线程修改的状态对另一个线程是可见的。volatile修饰可以保证可见性，它会保证修改的值会立即被更新到主存，所以对其他线程是可见的，普通的共享变量不能保证可见性，因为被修改后不会立即写入主存，何时被写入主存是不确定的，所以其他线程去读取的时候可能读到的还是旧值

有序性：Java中的指令重排序（包括编译器重排序和运行期重排序）可以起到优化代码的作用，但是在多线程中会影响到并发执行的正确性，使用volatile可以保证有序性，禁止指令重排

volatile可以保证可见性 有序性，但是无法保证原子性，在某些情况下可以提供优于锁的性能和伸缩性，替代sychronized关键字简化代码，但是要严格遵循使用条件。

### 11，谈谈对Synchronized关键字，类锁，方法锁，重入锁的理解

java的对象锁和类锁：java的对象锁和类锁在锁的概念上基本上和内置锁是一致的，但是，两个锁实际是有很大的区别的，

对象锁是用于对象实例方法，或者一个对象实例上的，

类锁是用于类的静态方法或者一个类的class对象上的。

我们知道，类的对象实例可以有很多个，但是每个类只有一个class对象，所以不同对象实例的对象锁是互不干扰的，但是每个类只有一个类锁。但是有一点必须注意的是，其实类锁只是一个概念上的东西，并不是真实存在的，它只是用来帮助我们理解锁定实例方法和静态方法的区别的

### 12，synchronized和volatile比较

* synchronized关键字

对象锁；下面2个其实是等价的，都是锁住set方法

public synchronized void set(){} public void set(){synchronized (this){}}

synchronized和volatile的区别

声明为volatile的变量，

本质是在告诉jvm当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取；

从主内存获取数据，将在所有线程中同步内存数据，在任意一个线程总修改，其他线程都会同步更新该变量的值；会消耗性能

只能声明变量无法声明方法。仅能实现变量的修改可见性，不能保证原子性；不阻塞线程；被标记的变量不会被编译器优化。

volatile为实例域的同步访问提供了免锁机制，如果声明一个域为volatile，那么编译器和虚拟机就知道该域可能被另一个线程并发更新

声明为synchronized的

性能没有volatile好，因为它有一个监听器，更改后就会通知锁定的线程

锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住；当前线程更新后会同步其他所有线程，主要是因为同步主内存的数据

可以使用在类，方法，变量上；可以保证变量的修改可见性和原子性；会阻塞线程；标记的变量可以被编译器优化。

### 13、ReentrantLock 、synchronized和volatile比较

Java在过去很长一段时间只能通过synchronized关键字来实现互斥，它有一些缺点。

比如你不能扩展锁之外的方法或者块边界，尝试获取锁时不能中途取消等。

Java 5 通过Lock接口提供了更复杂的控制来解决这些问题。

ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义且它还具有可扩展性。

### 14，什么是线程池，如何使用?

创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。

从JDK1.5开始，Java API提供了Executor框架让你可以创建不同的线程池。

比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。

好处：降低资源消耗，提高响应速度，提高线程的可管理性

线程间操作LIst：List list = Collections.synchronizedList(new ArrayList());

扩展：有三个线程T1，T2，T3，怎么确保它们按顺序执行？

在多线程中有多种方法让线程按特定顺序执行，你可以用线程类的join()方法在一个线程中启动另一个线程，另外一个线程完成该线程继续执行。为了确保三个线程的顺序你应该先启动最后一个(T3调用T2，T2调用T1)，这样T1就会先完成而T3最后完成。

### 15，死锁的四个必要条件？

死锁产生的原因

1. 系统资源的竞争：系统资源的竞争导致系统资源不足，以及资源分配不当，导致死锁。

2. 进程运行推进顺序不合适

* 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。此时若有其他进程请求该资源，则请求进程只能等待。
* 请求与保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源 已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。
* 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，即只能 由获得该资源的进程自己来释放（只能是主动释放)。
* 循环等待条件: 若干进程间形成首尾相接循环等待资源的关系

这四个条件是死锁的必要条件，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。

死锁的避免与预防：

死锁避免的基本思想:

系统对进程发出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,如果分配后系统可能发生死锁,则不予分配,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。

理解了死锁的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和解除死锁。所以，在系统设计、进程调度等方面注意如何让这四个必要条件不成立，如何确定资源的合理分配算法，避免进程永久占据系统资源。此外，也要防止进程在处于等待状态的情况下占用资源。因此，对资源的分配要给予合理的规划。

死锁避免和死锁预防的区别：

死锁预防是设法至少破坏产生死锁的四个必要条件之一,严格的防止死锁的出现,而死锁避免则不那么严格的限制产生死锁的必要条件的存在,因为即使死锁的必要条件存在,也不一定发生死锁。死锁避免是在系统运行过程中注意避免死锁的最终发生。

### 16，Java虚拟机的特性：

一般的高级语言要在不同的平台运行，至少需要编译成不同的目标代码。而Java在不同平台运行时不需要重新编译，虚拟机屏蔽具体平台相关信息，只需要生成在Java虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改的运行，Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。

### 17，实现一个 Json 解析器(可以通过正则提高速度)

String json = "{name:\"jason\",father:\"jason\",age:18}";
//name:"jason"
//age:18
//\"\\w+\" 字符串属性
Pattern p = Pattern.compile("\\w+:(\"\\w+\"|\\d\*)");
Matcher m = p.matcher(json);
while(m.find()){
 String text = m.group();
 int dotPos= text.indexOf(":");
 String key = text.substring(0, dotPos);
 String value = text.substring(dotPos+1, text.length());
 //替换字符串的开始结束的双引号
 value = value.replaceAll("^\\\"|\\\"$", "");
 System.out.println(key);
 System.out.println(value);
}

### 18，讲一下常见编码方式？

编码的意义：计算机中存储的最小单元是一个字节即8bit，所能表示的字符范围是255个，而人类要表示的符号太多，无法用一个字节来完全表示，固需要将符号编码，将各种语言翻译成计算机能懂的语言。

1）ASCII码：总共128个，用一个字节的低7位表示，0〜31控制字符如换回车删除等；32~126是打印字符，可通过键盘输入并显示出来；

2）ISO-8859-1,用来扩展ASCII编码，256个字符，涵盖了大多数西欧语言字符。

3）GB2312:双字节编码，总编码范围是A1-A7,A1-A9是符号区，包含682个字符，B0-B7是汉字区，包含6763个汉字；

4）GBK为了扩展GB2312,加入了更多的汉字，编码范围是8140~FEFE，有23940个码位，能表示21003个汉字。

5）UTF-16: ISO试图想创建一个全新的超语言字典，世界上所有语言都可通过这本字典Unicode来相互翻译，而UTF-16定义了Unicode字符在计算机中存取方法，用两个字节来表示Unicode转化格式。不论什么字符都可用两字节表示，即16bit，固叫UTF-16。

6）UTF-8：UTF-16统一采用两字节表示一个字符，但有些字符只用一个字节就可表示，浪费存储空间，而UTF-8采用一种变长技术，每个编码区域有不同的字码长度。  不同类型的字符可以由1~6个字节组成。

### 19，LRUCache原理

LruCache是个泛型类，主要原理是：把最近使用的对象用强引用存储在LinkedHashMap中，当缓存满时，把最近最少使用的对象从内存中移除，并提供get/put方法完成缓存的获取和添加。LruCache是线程安全的，因为使用了synchronized关键字。

当调用put()方法，将元素加到链表头，如果链表中没有该元素，大小不变，如果没有，需调用trimToSize方法判断是否超过最大缓存量，trimToSize()方法中有一个while(true)死循环，如果缓存大小大于最大的缓存值,会不断删除LinkedHashMap中队尾的元素，即最少访问的，直到缓存大小小于最大缓存值。当调用LruCache的get方法时，LinkedHashMap会调用recordAccess方法将此元素加到链表头部。

### 20，新问题，面试的时候碰到的，Java怎么跳出多重循环？

在java中，使用break可以跳出循环，默认情况下是跳出最里层的循环，假如我们要跳出多层循环怎么办呢，Java替我们已经做好了这一点，就是用 循环标签 ：即是对某个循环定义一个名字，然后在 break 后面加上这个名字，当符合 break 条件时，程序就会跳到规定的循环，当然其实还有另外2种方法，总共是有三种 第一种，使用带有标号的的break语句 第二种，外层的循环条件收到内层的代码控制限制 第三种，使用try/catch强制跳出循环 参考：https://blog.csdn.net/qq\_37107280/article/details/73556419?utm\_medium=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase