1. **java访问修饰符public,private,protected,以及不写（默认）时的区别？**
2. **静态内部类和匿名内部类的区别**

静态内部类：**不需要一个外部类的实例为基础**，可以直接实例化  
语法：**new 外部类.静态内部类();**  
因为没有一个外部类的实例，所以在静态内部类里面**不可以访问外部类的实例属性和方法**  
除了可以访问外部类的**私有静态成员外**，静态内部类和普通类没什么大的区别

匿名内部类：指的是在**声明一个类的同时实例化它**

**应用场景**：有的时候，为了快速使用，直接实例化一个抽象类，并“**当场**”实现其抽象方法。既然实现了抽象方法，那么就是一个新的类，只是这个类，没有命名。这样的类，叫做匿名类

1. **java内部类调用外部类的方法**

直接使用外部类名称.this.方法即可，例如Inner是内部类，Outer是外部类，在Inner中调用Outer.this.Print()即可使用Outer的print方法

1. **内存泄漏与内存溢出关系**

**内存泄漏memory leak** **:**是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄漏似乎不会有大的影响，但内存泄漏堆积后的后果就是内存溢出。

**内存溢出 out of memory :**指程序申请内存时，没有足够的内存供申请者使用，或者说，给了你一块存储int类型数据的存储空间，但是你却存储long类型的数据，那么结果就是内存不够用，此时就会报错OOM,即所谓的内存溢出。

**关系：**

1. 内存泄漏的堆积最终会导致内存溢出
2. 内存溢出就是你要的内存空间超过了系统实际分配给你的空间，此时系统相当于没法满足你的需求，就会报内存溢出的错误
3. 内存泄漏是指你向系统申请分配内存进行使用(new)，可是使用完了以后却不归还(delete)，结果你申请到的那块内存你自己也不能再访问（也许你把它的地址给弄丢了），而系统也不能再次将它分配给需要的程序
4. 分配的内存不足以放下数据项序列

**内存溢出的解决方案：**  
第一步，修改JVM启动参数，直接增加内存。(-Xms，-Xmx参数一定不要忘记加。)

第二步，检查错误日志，查看“OutOfMemory”错误前是否有其 它异常或错误。

第三步，对代码进行走查和分析，找出可能发生内存溢出的位置。

* 1. **重点排查以下几点：**  
     检查对数据库查询中，是否有一次获得全部数据的查询。一般来说，如果一次取十万条记录到内存，就可能引起内存溢出。这个问题比较隐蔽，在上线前，数据库中数据较少，不容易出问题，上线后，数据库中数据多了，一次查询就有可能引起内存溢出。因此对于数据库查询尽量采用分页的方式查询。
  2. 检查代码中是否有死循环或递归调用。
  3. 检查是否有大循环重复产生新对象实体。
  4. 检查List、MAP等集合对象是否有使用完后，未清除的问题。List、MAP等集合对象会始终存有对对象的引用，使得这些对象不能被GC回收。

第四步，使用内存查看工具动态查看内存使用情况

1. **判断对象是否已死**

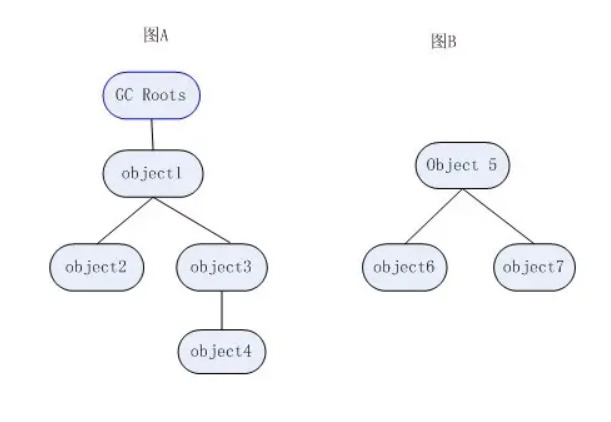
**一、引用计数法**

给对象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的对象就是不可能被再使用的。

主流的JVM里面没有选用引用计数算法来管理内存，其中最主要的原因是它很难解决对象间的互循环引用的问题。

**二、可达性分析算法**

通过一系列的称为“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（就是从GC Roots 到这个对象是不可达），则证明此对象是不可用的。所以它们会被判定为可回收对象（例如图B中的对象既是不可达的）。



在可达性分析算法中，要真正宣告一个对象死亡，至少要经历两次标记过程：

1.如果对象在进行可达性分析后发现没有与GC Roots相连接的引用链，那它将会被第一次标记并且进行一次筛选，筛选的条件是此对象是否有必要执行finalize()方法。当对象没有覆盖finalize()方法，或者finalize()方法已经被虚拟机调用过，虚拟机将这两种情况都视为“没有必要执行”。

2.如果这个对象被判定为有必要执行finalize()方法，那么这个对象将会放置在一个叫做F-Queue队列之中，并在稍后由一个由虚拟机自动建立的、低优先级的Finalizer线程去执行它。finalize()方法是对象逃脱死亡命运的最后一次机会，稍候GC将对F-Queue中的对象进行第二次小规模的标记，如果对象要在finalie()中成功拯救自己——只要重新与引用链上的任何一个对象建立关联即可，譬如把自己（this关键字）赋值给某个类变量或者对象的成员变量，那在第二次标记时它将会被移除出“即将回收”的集合；如果对象这时候还没有逃脱，那基本上它就真的被回收了。

**三、判断对象是否存活与“引用”有关**

在JDK1.2之后，Java对引用的概念进行了扩充，将引用分为强引用（Strong Reference）、软引用（Soft Reference）、弱引用（Weak Reference）、虚引用（Phantom Reference）四种，这四种引用强度依次逐渐减弱。

强引用：就是指在程序代码之中普遍存在的，类似“Object obj = new Object()”这类的引用，只要强引用还存在，垃圾收集器永远不会回收掉被引用的对象。

软引用：用来描述一些还有用但并非必须的对象。在系统将要发生内存溢出异常之前，将会把这些对象列进回收范围之中进行第二次回收。

弱引用：用户描述非必须对象的。被弱引用关联的对象只能生存到下一次垃圾收集发生之前。当垃圾收集器工作时，无论当前内存是否足够，都会回收掉只被弱引用关联的对象。

虚引用：一个对象是否有虚引用存在，完全不会对其生存时间构成影响，也无法通过虚引用来取得一个对象实例。为一个对象设置虚引用的唯一目的就是能在这个对象被收集器回收时刻得到一个系统通知。

1. **Java异常体系（运行时与非运行时异常的区别）**

运行时异常和非运行时异常

(1)运行时异常都是RuntimeException类及其子类异常，如NullPointerException、IndexOutOfBoundsException等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引起的，程序应该从逻辑角度尽可能避免这类异常的发生。

当出现RuntimeException的时候，我们可以不处理。当出现这样的异常时，总是由虚拟机接管。比如：我们从来没有人去处理过NullPointerException异常，它就是运行时异常，并且这种异常还是最常见的异常之一。

出现运行时异常后，如果没有捕获处理这个异常（即没有catch），系统会把异常一直往上层抛，一直到最上层，如果是多线程就由Thread.run()抛出，如果是单线程就被main()抛出。抛出之后，如果是线程，这个线程也就退出了。如果是主程序抛出的异常，那么这整个程序也就退出了。运行时异常是Exception的子类，也有一般异常的特点，是可以被catch块处理的。只不过往往我们不对他处理罢了。也就是说，你如果不对运行时异常进行处理，那么出现运行时异常之后，要么是线程中止，要么是主程序终止。

如果不想终止，则必须捕获所有的运行时异常，决不让这个处理线程退出。队列里面出现异常数据了，正常的处理应该是把异常数据舍弃，然后记录日志。不应该由于异常数据而影响下面对正常数据的处理。

1. 非运行时异常是RuntimeException以外的异常，类型上都属于Exception类及其子类。如IOException、SQLException等以及用户自定义的Exception异常。对于这种异常，JAVA编译器强制要求我们必需对出现的这些异常进行catch并处理，否则程序就不能编译通过。所以，面对这种异常不管我们是否愿意，只能自己去写一大堆catch块去处理可能的异常。
2. **“抽象类”和“接口”的区别**

1、抽象类和接口都不能直接实例化，如果要实例化，抽象类变量必须指向实现所有抽象方法的子类对象，接口变量必须指向实现所有接口方法的类对象。

2、抽象类要被子类继承，接口要被类实现。

3、接口只能做方法申明，抽象类中可以做方法申明，也可以做方法实现

4、接口里定义的变量只能是公共的静态的常量，抽象类中的变量是普通变量。

5、抽象类里的抽象方法必须全部被子类所实现，如果子类不能全部实现父类抽象方法，那么该子类只能是抽象类。同样，一个实现接口的时候，如不能全部实现接口方法，那么该类也只能为抽象类。

6、抽象方法只能申明，不能实现，接口是设计的结果 ，抽象类是重构的结果

7、抽象类里可以没有抽象方法

8、如果一个类里有抽象方法，那么这个类只能是抽象类

9、抽象方法要被实现，所以不能是静态的，也不能是私有的。

10、接口可继承接口，并可多继承接口，但类只能单根继承。

如果你拥有一些方法并且想让它们中的一些有默认实现，那么使用抽象类吧。

如果你想实现多重继承，那么你必须使用接口。由于Java不支持多继承，子类不能够继承多个类，但可以实现多个接口。因此你就可以使用接口来解决它。

如果基本功能在不断改变，那么就需要使用抽象类。如果不断改变基本功能并且使用接口，那么就需要改变所有实现了该接口的类。

**扩展资料：**

**抽象类：**

抽象类是用来捕捉子类的通用特性的 。它不能被实例化，只能被用作子类的超类。抽象类是被用来创建继承层级里子类的模板。

**接口：**

接口是抽象方法的集合。如果一个类实现了某个接口，那么它就继承了这个接口的抽象方法。这就像契约模式，如果实现了这个接口，那么就必须确保使用这些方法。接口只是一种形式，接口自身不能做任何事情。

1. **java中创建多线程的四种方式**

<https://www.cnblogs.com/benjieqiang/p/11376076.html>

* 继承Thread类
* 实现Runnable接口
* 实现Callable接口
* 使用线程池

1. **Java 并发之线程间通信协作**

<https://www.cnblogs.com/xdyixia/p/9386133.html>

* syncrhoized加锁的线程的Object类的wait()/notify()/notifyAll()
* ReentrantLock类加锁的线程的Condition类的await()/signal()/signalAll()
* 通过管道进行线程间通信：1）字节流；2）字符流

1. **有三种方法可以结束线程**

<https://blog.csdn.net/xu__cg/article/details/52831127>

<https://www.jianshu.com/p/70ce57c57065>

* 设置退出标志，使线程正常退出，也就是当run()方法完成后线程终止
* 使用interrupt()方法中断线程
* 使用stop方法强行终止线程（不推荐使用，Thread.stop, Thread.suspend, Thread.resume 和Runtime.runFinalizersOnExit 这些终止线程运行的方法已经被废弃，使用它们是极端不安全的！）

1. **数组和链表区别**

1、数组是一种线性表数据结构。它用一组连续的内存空间，来存储一组具有相同类型的数据。最大的特点就是支持随机访问，但插入、删除操作也因此变得比较低效,平均情况时间复杂度为O (n)。在平时的业务开发中，我们可以直接使用编程语言提供的容器类,但是，如果是特别底层的开发,直接使用数组可能会更合适。

2、链表它并不需要一块连续的内存空间，它通过"指针"将一组零散的内存，空间可扩容，比较常用的是单链表,双链表和循环链表。和数组相比，链表更适合插入、删除操作频繁的场景，查询的时间复杂度较高。不过,在具体软件开发中，要对数组和链表的各种性能进行对比，综合来选择使用两者中的哪一个。

1. **ThreadLocal是什么**

<https://www.cnblogs.com/cjsblog/p/9773079.html>

* ThreadLocal是一种变量类型，我们称之为“线程局部变量”
* 每个线程访问这种变量的时候都会创建该变量的副本，这个变量副本为线程私有
* ThreadLocal类型的变量一般用private static加以修饰

1. **java线程同步的实现方式**

<https://blog.csdn.net/qq_34147021/article/details/88325629>

<https://www.cnblogs.com/xiaoxi/p/7679470.html>

* 使用 synchronized关键字
* **使用特殊域变量(volatile)实现线程同步**
* **使用重入锁实现线程同步（**ReentrantLock**）**
* **使用局部变量实现线程同步**
* **使用阻塞队列实现线程同步**
* **使用原子变量实现线程同步**

1. **Java三大特性（封装，继承，多态）**

* 封装指的是属性私有化，根据需要提供setter和getter方法来访问属性。即隐藏具体属性和实现细节，仅对外开放接口，控制程序中属性的访问级别。封装目的是增强安全性和简化编程，使用者不必在意具体实现细节，而只是通过外部接口即可访问类的成员。
* 继承是指将多个相同的属性和方法提取出来，新建一个父类。Java中一个类只能继承一个父类，且只能继承访问权限非private的属性和方法。 子类可以重写父类中的方法，命名与父类中同名的属性。继承目的是代码复用。
* 多态可以分为两种：设计时多态和运行时多态。

设计时多态：即重载，是指Java允许方法名相同而参数不同（返回值可以相同也可以不相同）。

运行时多态：即重写，是指Java运行根据调用该方法的类型决定调用哪个方法。

多态目的：增加代码的灵活度。

1. **Java中的final关键字**

<https://www.cnblogs.com/xuelisheng/p/11158110.html>

* 当用final修饰一个类时，表明这个类不能被继承
* 使用final方法的原因有两个。第一个原因是把方法锁定，以防任何继承类修改它的含义；第二个原因是效率。
* final成员变量表示常量，只能被赋值一次，赋值后值不再改变

1. **volatile有什么作用**

<https://www.jianshu.com/p/933946794341>

<https://blog.csdn.net/hxpjava1/article/details/55188908>

* 状态标志
* 一次性安全发布（one-time safe publication）
* 独立观察（independent observation）
* “volatile bean” 模式
* 开销较低的“读－写锁”策略

1. **进程,线程,协程区别**

<https://www.cnblogs.com/fanguangdexiaoyuer/p/10834737.html>

1. 进程是资源分配的单位

2. 线程是操作系统调度的单位

3. 进程切换需要的资源很最大，效率很低

4. 线程切换需要的资源一般，效率一般

5. 协程切换任务资源很小，效率高

6. 多进程、多线程根据cpu核数不一样可能是并行的 也可能是并发的。协程的本质就是使用当前进程在不同的函数代码中切换执行，可以理解为并行。协程是一个用户层面的概念，不同协程的模型实现可能是单线程也可能是多线程。

1. **wait不在同步块里面执行会怎么样**

<https://www.cnblogs.com/set-cookie/p/8686218.html>

使用wait和notify的话，那么必须在synchronized块中，否则会抛出IllegalMonitorStateException

1. **虚拟内存和物理内存有什么区别？**

物理内存即你电脑主板内存槽上的实际容量。虚拟内存是当物理内存不够用时系统自动从硬盘空间划分一些容量出来当做虚拟内存使用（即辅助物理内存）

1. **static用法**

<https://www.cnblogs.com/AlmostWasteTime/p/8757621.html#_label1>

* 修饰成员变量

当用static修饰时，例如static String name，这个变量是存在静态区，被堆中的对象所共享

* 修饰成员方法

static修饰方法后，最大的作用就是可以"类名.方法名"调用该方法

* 静态块

static修饰静态块后，将要修饰的成员同一放在一个static中。该静态化会在第一次使用类时，最先初始化static修饰的静态块。并且在程序运行过程中，只需要初始化一次，不会进行多次初始化。

* 静态导包

静态导包指在导入包前用static修饰，作用是当使用静态导包后，引用导入类的方法（变量）时，可以不用XXX=类名.方法，直接XXX=方法（）。

1. **synchronized修饰静态方法和普通方法有什么区别**

<https://blog.csdn.net/qq_35181209/article/details/74838690>

在static方法前加synchronized：静态方法属于类方法，它属于这个类，获取到的锁，是属于类的锁。

在普通方法前加synchronized：非static方法获取到的锁，是属于当前对象的锁。

1. **单向链表和双向链表有什么区别？各自有什么优缺点？**

<https://zhidao.baidu.com/question/225218516.html>

一、指代不同

1、双向链表：也叫双链表，是链表的一种，每个数据结点中都有两个指针，分别指向直接后继和直接前驱

2、单向链表：是链表的一种，其特点是链表的链接方向是单向的，对链表的访问要通过顺序读取从头部开始。

二、优点不同

1、双向链表：从双向链表中的任意一个结点开始，都可以很方便地访问前驱结点和后继结点。

2、单向链表：单个结点创建非常方便，普通的线性内存通常在创建的时候就需要设定数据的大小，结点的访问方便，可以通过循环或者递归的方法访问到任意数据。

三、缺点不同

1、双向链表：增加删除节点复杂，需要多分配一个指针存储空间。

2、单向链表：结点的删除非常方便，不需要像线性结构那样移动剩下的数据，但是平均的访问效率低于线性表。

1. **Parcelable和Serializable的区别**

<https://blog.csdn.net/m_wbcg/article/details/70766976>

* Serializable, 简单易用
* 缺点是使用了反射，序列化的过程较慢。这种机制会在序列化的时候创建许多的临时对象，容易触发垃圾回收
* Parcelable, 速度至上
* 实现Parcelable并不容易。实现Parcelable接口需要写大量的模板代码，这使得对象代码变得难以阅读和维护

1. **Java反射以及在安卓中的应用**

<https://juejin.im/post/5deb825ee51d4557fa60e416>

Java反射是指，在运行状态中，对于任意一个类，都能知道这个类的所有属性及方法，对于任何一个对象，都能调用他的任何一个方法和属性，这种动态获取新的对象及动态调用对象的方法的功能叫做反射.

应用：

Android 中 StorageManager 类中的 getVolumePaths() 方法，该方法为隐藏方法，没办法正常调用，但是在实际使用中我们也可能用上，如果你有系统权限，那你就可以像 Android SD卡及U盘插拔状态监听及内容读取这样为所欲为

1. **Java中如何判断两个对象是否相等**

<https://blog.csdn.net/u013063153/article/details/78808923>

一、java中的基本数据类型（int、double、short、long、byte、float、boolean、char）判断是否相等，直接使用"=="就行了，相等返回true，否则，返回false。

二、但是java中的引用类型的对象比较变态，假设有两个引用对象obj1,obj2,

obj1==obj2 判断是obj1,obj2这两个引用变量是否相等，即它们所指向的对象是否为同一个对象。言外之意就是要求两个变量所指内存地址相等的时候，才能返回true，每个对象都有自己的一块内存，因此必须指向同一个对象才返回ture。

三、如果想要自定义两个对象（不是一个对象，即这两个对象分别有自己的一块内存）是否相等的规则，那么必须在对象的类定义中重写equals()方法，如果不重写equals()方法的话，默认的比较方式是比较两个对象是否为同一个对象。

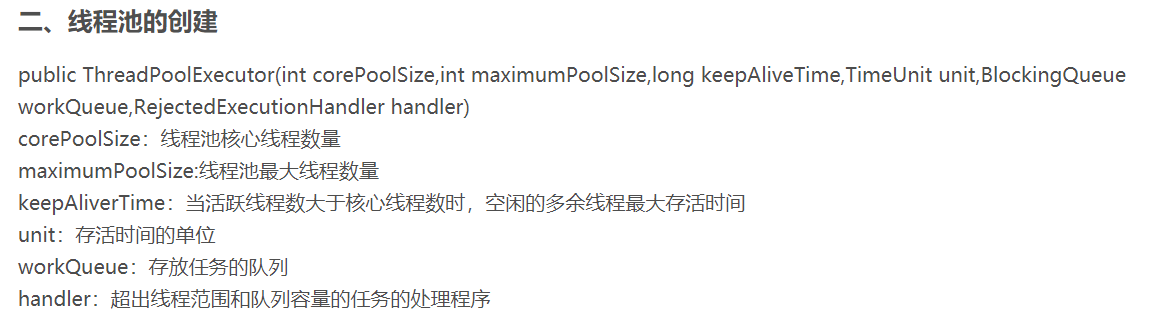
在Java API中，有些类重写了equals()方法，它们的比较规则是：当且仅当该equals方法参数不是 null，两个变量的类型、内容都相同，则比较结果为true。这些类包括：String、Double、Float、Long、Integer、Short、Byte、、Boolean、BigDecimal、BigInteger等等，太多太多了，但是常见的就这些了，具体可以查看API中类的equals()方法，就知道了。

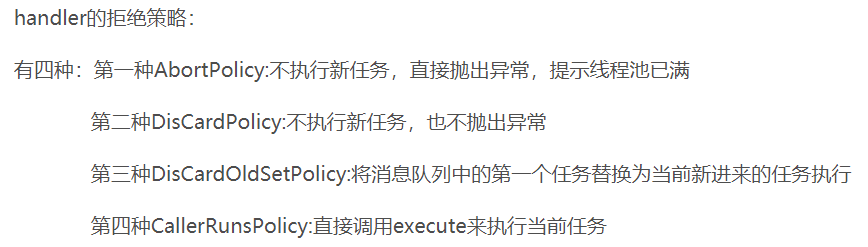
1. **Java线程池**

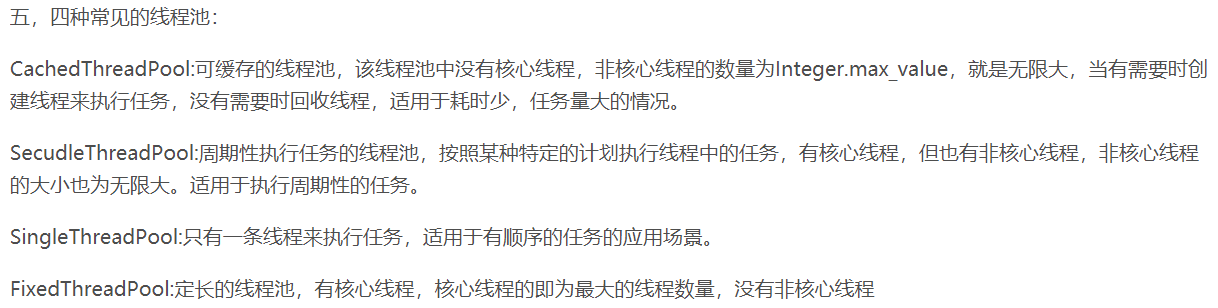
<https://blog.csdn.net/qq_22764659/article/details/83620730>

<https://www.jianshu.com/p/9beab78a3afe>

<https://blog.csdn.net/weixin_40271838/article/details/79998327>







1. **Java线程池常见问题**

<https://blog.csdn.net/xiaobao5214/article/details/88309552>

Q. 线程池是什么时候创建线程的？

A.任务提交的时候

Q.任务runnable task是先放到core到maxThread之间的线程，还是先放到队列？

A.先放队列!!!

Q. 队列中的任务是什么时候取出来的？

A. worker中 runWorker() 一个任务完成后，会取下一个任务

1. **CountDownLatch原理**

<https://www.cnblogs.com/xwb583312435/p/9007216.html>

CountDownLatch是在java1.5被引入的，跟它一起被引入的并发工具类还有CyclicBarrier、Semaphore、ConcurrentHashMap和BlockingQueue，它们都存在于java.util.concurrent包下。CountDownLatch这个类能够使一个线程等待其他线程完成各自的工作后再执行。

CountDownLatch是通过一个计数器来实现的，计数器的初始值为线程的数量。每当一个线程完成了自己的任务后，计数器的值就会减1。当计数器值到达0时，它表示所有的线程已经完成了任务，然后在闭锁上等待的线程就可以恢复执行任务。

**实时系统中的使用场景：**

* **实现最大的并行性**
* **开始执行前等待n个线程完成各自任务**
* **死锁检测**

1. **Thread和Runnable有什么区别?**

<https://www.cnblogs.com/J-Simon/p/11470758.html>

* Runnable的实现方式是实现其接口即可
* Thread的实现方式是继承其类
* Runnable接口支持多继承，但基本上用不到
* Thread实现了Runnable接口并进行了扩展，而Thread和Runnable的实质是实现的关系

1. **String，StringBuffer，StringBuilder有什么区别？**

* String类是不可变的，所谓不可变意思就是创建一个类后任何对String的改变都 会引发新的String对象的生成；
* StringBuffer不同于String的是StringBuffer是可变的
* StringBuffer支持并发操作，线性安全的，适合多线程中使用
* StringBuilder不支持并发操作，线性不安全的，不适合多线程中使用，但其在单线程中的性能比StringBuffer高。

1. **类加载机制**

<https://www.cnblogs.com/jsersudo/p/10179082.html>

类的加载可以分为加载、连接、初始化 三个步骤

类的加载：

1、类的加载指的是将类的class文件加载到内存中，并且为之创建一个java.lang.Class对象。

2、JVM提供类加载器去加载类。JVM提供的类加载器通常称为系统类加载器，可以继承ClassLoader基类去创建自己的类加载器。

3、程序中所有的类实际上也是实例，他们都是java.lang.Class实例。

4、使用不同的类加载器可以从不同来源来加载类的二进制数据（本地文件系统、JAR包、网络、java源文件动态编译进行加载）。

类的连接：

1、连接阶段负责把类的二进制文件数据合并到JRE中。

2、可以分为三个阶段：

验证：验证类结构，是否和其他类协调一致。

准备：为类的类变量分配内存，设置默认初始值。

解析：将类的二进制文件中的符号引用替换为直接引用。

类的初始化：

1、主要对类变量初始化，（声明类变量时指定初始值、使用静态初始化块）。

2、步骤：

当这个类没有加载和连接，则先加载并且连接。

当这个类的直接父类没有被初始化，则先初始化。

当类中有初始化语句，则系统依次执行初始化语句。

1. **引用和指针的区别**

指针和引用的联系与区别

★ 相同点：

1. 都是地址的概念；

指针指向一块内存，它的内容是所指内存的地址；引用是某块内存的别名。

★ 区别：

1. 指针是一个实体，而引用仅是个别名；

2. 引用使用时无需解引用(\*)，指针需要解引用；

3. 引用只能在定义时被初始化一次，之后不可变；指针可变；

4. 引用没有 const，指针有 const；

5. 引用不能为空，指针可以为空；

6. “sizeof 引用”得到的是所指向的变量(对象)的大小，而“sizeof 指针”得到的是指针本身(所指向的变量或对象的地址)的大小；

7. 指针和引用的自增(++)运算意义不一样；

8.从内存分配上看：程序为指针变量分配内存区域，而引用不需要分配内存区域。

1. **单核多线程与多核多线程**

<https://blog.csdn.net/qq_27185561/article/details/82900426>

单核多线程指的是单核CPU轮流执行多个线程，通过给每个线程分配CPU时间片来实现，只是因为这个时间片非常短（几十毫秒），所以在用户角度上感觉是多个线程同时执行。

多核多线程，可以把多线程分配给不同的核心处理，其他的线程依旧等待，相当于多个线程并行的在执行，而单核多线程只能是并发。

* 单CPU中进程只能是并发，多CPU计算机中进程可以并行。
* 单CPU单核中线程只能并发，单CPU多核中线程可以并行。
* 无论是并发还是并行，使用者来看，看到的是多进程，多线程。

1. **Java中常见流的分类及简单讲解**

<https://www.cnblogs.com/xiohao/p/8997401.html>

1. **进程间怎么通信**

<https://blog.csdn.net/wh_sjc/article/details/70283843>

IPC的方式通常有管道（包括无名管道和命名管道）、消息队列、信号量、共享存储、Socket、Streams等。其中 Socket和Streams支持不同主机上的两个进程IPC

1. **死锁产生的条件，怎么解决？**

<https://www.cnblogs.com/loren-Yang/p/7577373.html>

产生死锁的四个必要条件：

（1）互斥条件：一个资源每次只能被一个进程使用。

（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

（3）不可剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。

（4）环路等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

1. **Java中的锁**

<https://www.cnblogs.com/jyroy/p/11365935.html>

Monitor可以理解为一个同步工具或一种同步机制，通常被描述为一个对象。每一个Java对象就有一把看不见的锁，称为内部锁或者Monitor锁。

Monitor是线程私有的数据结构，每一个线程都有一个可用monitor record列表，同时还有一个全局的可用列表。每一个被锁住的对象都会和一个monitor关联，同时monitor中有一个Owner字段存放拥有该锁的线程的唯一标识，表示该锁被这个线程占用。

现在话题回到synchronized，synchronized通过Monitor来实现线程同步，Monitor是依赖于底层的操作系统的Mutex Lock（互斥锁）来实现的线程同步。



1. **Reentrantlock和sychronized有什么区别**

<https://blog.csdn.net/qq_40551367/article/details/89414446>

1. **ReentrantReadWriteLock读写锁**

<https://www.cnblogs.com/wait-pigblog/p/9350569.html>

读写锁是一种特殊的自旋锁，它把对共享资源对访问者划分成了读者和写者，读者只对共享资源进行访问，写者则是对共享资源进行写操作。读写锁在ReentrantLock上进行了拓展使得该锁更适合读操作远远大于写操作的场景。一个读写锁同时只能存在一个写锁但是可以存在多个读锁，但不能同时存在写锁和读锁

1. **Java中List和Map的区别**

<https://www.sohu.com/a/166443991_99972329>

<https://www.cnblogs.com/ArvinSnow/p/11045095.html>

1. **hashmap hashtable**

<https://blog.csdn.net/xuhuaabc/article/details/91475761>

1、hashMap并不是在链表元素个数大于8就一定会转换为红黑树，而是先考虑扩容，扩容达到默认限制后才转换。  
2、hashMap的红黑树不一定小于6的时候才会转换为链表，而是只有在resize的时候才会根据 UNTREEIFY\_THRESHOLD 进行转换。

1. **深入浅出学Java——HashMap**

<https://blog.csdn.net/woshimaxiao1/article/details/83661464>

1. **Java的ConcurrentHashMap**

ConcurrentHashMap是Java中的一个线程安全且高效的HashMap实现。

1. **Java中Map的4种遍历方式**

<https://www.jianshu.com/p/323e272e74da>

* 平常用的最多也最可取的一种遍历方式entrySet

for (Map.Entry<String, Object> entry : map.entrySet()) {

log.info("key :" + entry.getKey() + " " + "value " + entry.getValue());

}

* 如果只需要map中的键或者值，你可以通过keySet或values来实现遍历

for (String key : map.keySet()) {

log.info("key :" + key + " " + "value " + map.get(key));

}

* 使用Iterator进行遍历Map

Iterator<Map.Entry<String, Object>> iterator = map.entrySet().iterator();

while (iterator.hasNext()) {

Map.Entry<String, Object> entry = iterator.next();

log.info("key " + entry.getKey() + " " + "value " + entry.getValue());

}

* 遍历value

for (Object value : map.values()) {

log.info("value: " + value);

}

1. **Map的线程安全？读多写少选哪个集合？**

方式1.  使用Hashtable

其实现原理是在增删改查的方法上使用了synchronized锁机制，在多线程环境下，无论是读数据，还是修改数据，在同一时刻只能有一个线程在执行synchronize方法，因为是对整个表进行锁定。所以线程越多，对该map的竞争越激烈，效率越低，不推荐使用。

方式2.  使用Collections.synchronizedMap(new Hashtable())

其实现原理是使用工具类里面的静态方法，把传入进来的Hashtable包装成同步的，即在增删改查的方法上增加了synchronized锁机制，其实现方式与Hashtable差不多，效率也差不多，不推荐使用

方式3.  使用ConcurrentHashMap

其实现原理是Hashtable是对整个表进行加锁，而ConcurrentHashMap是把表进行分段，初始情况下分成16段，每一段都有一把锁，当多个线程访问不同的段时，因为获取到的锁是不同的，所以可以并行的访问。效率比Hashtable高多了，推荐使用



1. **Java集合**

<https://www.cnblogs.com/lixiansheng/p/11348050.html>

<https://blog.csdn.net/feiyanaffection/article/details/81394745>

1. **java 泛型详解**

<https://www.cnblogs.com/ffaiss/p/11460555.html>

泛型擦除：

擦除是指编译为字节码后泛型信息被抹除了，这导致无法在运行时判断泛型的具体类型，同时继承关系也失效了（即泛型不变）。  
为了解决泛型不变的问题就引入了通配符？和限定符（extend、super），extend使得泛型协变（告诉编译器限定了上界，但代价是只能输出到父类，子类由于可以是任意的，编译器仍然无法推断子类输入是否合法，所以常用的场景就是作为方法返回或者成员变量输出），super使得泛型逆变（限定下界，可以输入子类，当然也可以输出到object，但是类型信息已经丢失，毫无意义，常用场景就是作为参数输入）

1. **Java单例模式的不同写法（懒汉式、饿汉式、双检锁、静态内部类、枚举）**

<https://blog.csdn.net/fly910905/article/details/79286680>

* 饿汉式：线程安全
* 懒汉式：非线程安全
* 双检锁：线程安全
* 静态内部类：线程安全
* 枚举：线程安全

1. **final, finally, finalize的区别。**

final 用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。finally是异常处理语句结构的一部分，表示总是执行。finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。

1. **sleep() 和 wait() 有什么区别?**

sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用sleep不会释放对象锁。wait是Object类的方法，对此对象调用wait方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出notify方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。

1. **error和exception有什么区别?**

error 表示恢复不是不可能但很困难的情况下的一种严重问题。比如说内存溢出。不可能指望程序能处理这样的情况。 exception 表示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。

1. **同步和异步有何异同，在什么情况下分别使用他们？举例说明。**

如果数据将在线程间共享。例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行同步存取。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。

1. **什么时候用assertion**

assertion(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。在实现中，

assertion就是在程序中的一条语句，它对一个boolean表达式进行检查，一个正确程序必须保证这个boolean表达式的值为true；如果该值为false，说明程序已经处于不正确的状态下，系统将给出警告或退出。一般来说，assertion用于保证程序最基本、关键的正确性。assertion检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion检查通常是关闭的。

1. **short s1 = 1; s1 = s1 + 1;有什么错? short s1 = 1; s1 += 1;有什么错?**

short s1 = 1; s1 = s1 + 1; （s1+1运算结果是int型，需要强制转换类型）

short s1 = 1; s1 += 1;（可以正确编译）

1. **Math.round(11.5)等於多少?**

Math.round(11.5)= 12

Math.round(-11.5)= -11

round方法返回与参数最接近的长整数，参数加1/2后求其floor.

1. **启动一个线程是用run()还是start()?**

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()方法可以产生必须退出的标志来停止一个线程。

1. **接口是否可继承接口? 抽象类是否可实现(implements)接口? 抽象类是否可继承实体类(concrete class)?**

接口可以继承接口。抽象类可以实现(implements)接口，抽象类是否可继承实体类的前提是实体类必须有明确的构造函数。

1. **当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?**

是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的。

1. **Java中的异常处理机制的简单原理和应用。**

当JAVA程序违反了JAVA的语义规则时，JAVA虚拟机就会将发生的错误表示为一个异常。违反语义规则包括2种情况。一种是JAVA类库内置的语义检查。例如数组下标越界,会引发IndexOutOfBoundsException;访问null的对象时会引发NullPointerException。另一种情况就是JAVA允许程序员扩展这种语义检查，程序员可以创建自己的异常，并自由选择在何时用throw关键字引发异常。所有的异常都是java.lang.Thowable的子类。

1. **什么是java序列化**

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。序列化是为了解决在对对象流进行读写操作时所引发的问题。

1. **写clone()方法时，通常都有一行代码，是什么？**

Clone 有缺省行为，super.clone();他负责产生正确大小的空间，并逐位复制。

1. **Collection 和 Collections的区别**

Collection是集合类的上级接口，继承与他的接口主要有Set 和List. Collections是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作

1. **float f=3.4;是否正确？**

不正确。3.4是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换float f =(float)3.4; 或者写成float f =3.4F;

1. **进程间通信**

管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

信号 ( signal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

套接字( socket ) ： 套接字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

1. **java7后关键字 switch 支不支持字符串作为条件**

switch(exp)，在JDK7之前，只能是byte、short、char、int或者对应的包装类，或者枚举常量（内部也是由整型或字符类型实现）。

为什么必须是这些呢，因为其实exp只是对int型支持的，其他都是因为可以自动拆卸或者自动向上转型到int，所以才可以。

到了JDK7的时候，String被引入了，为什么String能被引入呢？

其实本质上还是对int类型值得匹配。

原理如下，通过对case后面得String对象调用hashCode方法，得到一个int类型得hash值，然后用这个hash值来唯一标识这个case。那么当匹配时，首先调用exp的hashCode，得到exp的hash值，用这个hash值来匹配所有case，如果没有匹配成功，就说明不存在；如果匹配成功了，接着会调用字符串的equals方法进行匹配。（hash值一致，equals可不一定返回的就是true）。

所以，exp不能为null，cas子句使用的字符串也不能为null，不然会出现空指针异常。

1. **内部类**
2. 静态内部类：

静态内部类本身可以访问外部的静态资源，包括静态私有资源。但是不能访问非静态资源，可以不依赖外部类实例而实例化。

2. 成员内部类（在类中的内部类）：

成员内部类本身可以访问外部的所有资源，但是自身不能定义静态资源，因为其实例化本身就还依赖着外部类。

3. 局部内部类（在方法中的内部类）：

1. 局部内部类就像一个局部方法，不能被访问修饰符修饰，也不能被static修饰。

2. 局部内部类只能访问所在代码块或者方法中被定义为final的局部变量。

4. 匿名内部类：

1. 没有类名的内部类，不能使用class，extends和implements，没有构造方法。

2. 多用于GUI中的事件处理。

3. 不能定义静态资源

4. 只能创建一个匿名内部类实例。

5. 一个匿名内部类一定是在new后面的，这个匿名类必须继承一个父类或者实现一个接口。

6. 匿名内部类是局部内部类的特殊形式，所以局部内部类的所有限制对匿名内部类也有效。

1. **静态内部类实现的单例模式是如何保证线程安全的**

在类加载过程中，虚拟机会保证一个类的构造器方法在多线程环境中被正确地加载，同步，如果多个线程同时去初始化一个类，那么只有一个线程去执行这个类的构造器方法，其他线程都需要阻塞等待，直到活动线程执行方法完毕。

1. **双亲委派模式**

如果一个类加载器收到了类加载请求，它并不会自己先去加载，而是把这个请求委托给父类的加载器去执行，如果父类加载器还存在其父类加载器，则进一步向上委托，依次递归，请求最终将到达顶层的启动类加载器，如果父类加载器可以完成类加载任务，就成功返回，倘若父类加载器无法完成此加载任务，子加载器才会尝试自己去加载

1. **70**

数据库的共享锁，为什么这样设计

MVP和MVVM讲一下

操作系统：

<https://blog.csdn.net/Yasin0/article/details/81879569>

虚拟内存：

虚拟内存的目的是为了让物理内存扩充成更大的逻辑内存，从而让程序获得更多的可用内存。