**Override 和 Overload的区别**：

|  |
| --- |
| Override:跟面向对象有关，是多态的具体体现，注意被覆盖的函数名称，参数类型，返回类型都相同，简单的说就是子类重写了父类的方法。  Overload: 跟面向对象无关，是指编写一个与已有函数同名但是参数列表不一样的函数，其是一种语法规则 |

**进程和线程的区别**：

|  |
| --- |
| 1: 不同的操作系统资源管理方式  进程是系统进行资源分配和调度的一个独立单位，线程是进程的一个实体,是CPU调度和分派的基本单位  2: 内存占用的不同：  进程有独立的地址空间，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间。  3:效率问题：  进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率  4:健壮型问题：  多进程的程序要比多线程的程序健壮，因为单独的内存单元，不容易引起并发性的操作  5:总之：  线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源的管理和保护；而进程正相反。同时，线程适合于在SMP机器上运行，而进程则可以跨机器迁移。  6:形象的例子：（工厂和车间的例子） |

**实现一个栈(stack)**:用数组

|  |
| --- |
| 1: 底层用数组来实现，构造函数包括有参和无参的构造函数  2: 类的成员变量包括，Object[] arr, int top, int maxSize; => 注意数组的类型是object,表示什么类型都能装  3: 实现pop,push的操作都是基于top成员变量的设计，  pop() => top-1  push()=> top+1  但是要注意边界条件的检测，初始化的时候，top为-1. |

**实现一个队列(queue)**:用数组

|  |
| --- |
| 思想与上面类似，不过需要两个index来标记：  front 和 rear,代表队列的第一个元素和最后一个元素，当然操作也复杂一点，  例如超过maxSize的情况下，我们考虑用mod来处理：  this.rear = (this.rear+1) % this.maxSize; => 入队列  this.front = (this.front+1) % this.maxSize; =>出队列 |

**线程的五种状态和描述**:

|  |
| --- |
| **新建(New)：**new一个Thread对象或者其子类对象就是创建一个线程，当一个线程对象被创建，但是没有开启，这个时候，只是对象线程对象开辟了内存空间和初始化数据。  **就绪(Runnable)：**新建的对象调用start方法，就开启了线程，线程就到了就绪状态。       在这个状态的线程对象，具有执行资格，没有执行权。  **运行(Running)：**当线程对象获取到了CPU的资源。在这个状态的线程对象，既有执行资格，也有执行权。  **冻结(Blocked)：**运行过程中的线程由于某些原因(比如wait,sleep)，释放了执行资格和执行权。当然，他们可以回到运行状态。只不过，不是直接回到。 而是先回到就绪状态。  **死亡(Dead)：**当线程对象调用的run方法结束，或者直接调用stop方法，就让线程对象死亡，在内存中变成了垃圾。 |

**线程start()和run()方法的区别**:

|  |
| --- |
| 调用start方法方可启动线程，而run方法只是thread的一个普通方法，调用run方法不能实现多线程；具体的说：  **Start()方法:**  start方法用来启动线程,实现了多线程运行,这时无需等待run方法体代码执行完毕而直接继续执行下面的  代码。通过调用Thread类的start()方法来启动一个线程,这时此线程处于就绪(可运行)状态，并没有运行，  一旦得到cpu时间片(执行权),就开始执行run()方法,这里方法run()称为线程体，  它包含了要执行的这个线程的内容，Run方法运行结束,此线程随即终止。  **Run()方法:**  run()方法只是Thread类的一个普通方法,如果直接调用Run方法,程序中依然只有主线程这一个线程,  其程序执行路径还是只有一条，还是要等待run方法体执行完毕后才可继续执行下面的代码，  这样就没有达到多线程的目的 |

**线程sleep()和wait()方法的区别**:

|  |
| --- |
| (1)这两个方法来自不同的类，sleep()来自Thread类，和wait()来自Object类。  (2)sleep是Thread的静态类方法，谁调用的谁去睡觉，即使在a线程里调用了b的sleep方法，实际上还是a去睡觉， 要让b线程睡觉要在b的代码中调用sleep。而wait()是Object类的非静态方法  (3)sleep()释放资源不释放锁，而wait()释放资源释放锁；  (4)使用范围：wait,notify和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 |

**单例模式下的饿汉和懒汉式子以及懒汉式子的多线程表示**:

|  |
| --- |
| 饿汉：  class Single  {  private Single(){}//将构造函数私有化，不让别的类建立该类对象  private static final Single s=new Single();//自己建立一个对象  public static Single getInstance()//提供一个公共访问方式  {  return s;  }  }  懒汉线程不安全：  class Single  {  private Single(){}  private static Single s;  public static Single getInstance()  {  if(s==null)  s=new Single();  return s;  }  }  上述如果多个线程同时操作懒汉式时就有可能出现线程安全问题，解决线程安全问题可以加同步来解决。但是加了同步之后，每一次都要比较锁，效率就变慢了，  所以可以加**双重判断**来提高程序效率。  public static Single getInstance()  {  **if(s==null)//当不是null时候直接return**  {  synchronized(Single.class)  {  **if(s==null) //锁内部再判断一次，确保只有一个实例**  s=new Single();  }  }  return s;  } |

**线程中为什么wait()、notify()、notifyAll()**:

|  |
| --- |
| 为什么这些方法要定义在Object类中：  (1)这些方法只存在于同步中；  (2)使用这些方法时必须要指定所属的锁，即被哪个锁调用这些方法；  (3)而锁可以是任意对象，所以任意对象调用的方法就定义在Object中。  线程间的通信：  为什么要通信：多线程并发执行的时候， 如果需要指定线程等待或者唤醒指定线程， 那么就需要通信.比如生产者消费者的问题，  怎样通信：在同步代码块中, 使用锁对象的wait()方法可以让当前线程等待, 直到有其他线程唤醒为止.  使用锁对象的notify()方法可以唤醒一个等待的线程，或者notifyAll唤醒所有等待的线程.  多线程间通信用sleep很难实现，睡眠时间很难把握。 |

**线程中如果在静态方法使用同步什么后果：**

|  |
| --- |
| 同步静态方法时会获取该类的“Class”对象，所以当一个线程进入同步的静态方法中时，线程监视器获取类本身的对象锁，其它线程不能进入这个类的任何静态同步方法。它不像实例方法，因为多个线程可以同时访问不同实例同步实例方法。 |