# 二、多线程

windonws本身就是多进程的操作系统

进程：一个运行着的程序，在操作系统中通常用一个进程来表示。操作系统为这个进行分配资源执行程序。

线程：一个程序内部多个并行的执行线索，称为一个线程。一个进程内部可以开辟多个线程。多线程

例子：360 迅雷 JVM-垃圾回收器

## 1.开发线程的方法一

写一个类继承Thread，覆盖其中的run()方法，在run()方法中编写线程内部要执行的逻辑。

创建出该类的对象，调用start()方法既可开启一个新的线程。

所有的线程都来自Thread类，所以要创建一个线程就需要new一个Thread对象

继承Thread类覆盖run方法的目的，就是想要在run方法中指定线程要执行的代码

当new出该类的对象时，仅仅是在内存中创建出来了该类的一个对象而已，真正的线程并没有启动，需要调用start方法，开启线程，一旦调用这个方法，虚拟机就位这个线程开辟内存控制，分配cpu时间片，开始执行run中的代码，新的线程启动起来。

所谓的多个线程的并发执行其实是多个线程在抢夺cup的时间片，谁抢到谁就执行，严格来个每个线程执行都是断断续续的，只不过cpu时间片切换非常的迅速，在我们看起来就好像每个线程都是在并发着连续执行一样。

而多个线程对cpu时间片的抢夺，如果不做任何控制，完全是随机的，所以多线程并发执行时先后顺序没有规律，多次执行是执行顺序也会不相同。

这种方式创建线程缺点是，每次都创建新的线程对象具有各自的变量，无法共享数据。如果想要共享数据必须用静态，但是静态用多了不好，要慎用静态。

java只支持单继承，所以如果类已经有一个父类就不能用这种方式创建线程类。

案例：写一个程序模拟两个学生同时问问题

getName()

setName()

Thread.currentThread();//获取当前执行的线程

案例：铁路售票

## 2.创建线程的方式二

写一个类实现Runnable接口，实现其中要求的run方法，在run方法中写线程要执行的逻辑代码

new Thread过程中通过参数传入写好的Runnable接口实现类的对象，调用start方法启动线程

通过实现Runnable接口中的run方法，再通过创建Thread的过程中传入接口实现类的对象，告知Thread要执行的run的逻辑。

一旦调用start方法就分配内存和cpu时间片启动线程，执行run中的代码。

可以实现成员变量的共享

java中可以实现多接口，所以不会有单继承的问题

## 3.多线程并发安全

原因：多个线程并发操作共享的资源，操作过程中包含多步操作，由于线程的并发造成执行的混乱产生的问题。

解决方法1:使用同步代码块隔离多线程

同步代码块要将造成多线程并发安全问题的代码都要扩上。一般来说多线程并发安全问题通常都是由于同时执行读取和修改操作造成的问题，在使用同步代码块时，将读和改的操作都要括起来。

同步代码块可以解决多线程并发安全问题，但是在同步代码块中相当于将多线程变为了单线程，程序的性能会降低。

所以同步代码块中的代码不能太多，应该在保证能够防止多线程并发安全问题的前提下包裹的代码尽量的少，从而减少对程序性能的影响。

任何对象都可以当做锁对象来使用，但是必须保证这个对象是所有要用这个锁的并发线程都能看到的同一个对象才可以。多个线程用同一个锁对象，在它身上控制锁的开关，控制程序的并发。

常见的锁对象 ： 共享资源对象、this、类的class字节码对象

synchronized (obj) {

}

解决方法2:使用同步函数解决问题

如果发现某个方法整体上都需要同步可以将这个方法设置为同步方法，这个方法本身自动就会用锁隔离。

同步函数默认使用的锁对象是this

如果同步函数是静态方法，没有this对象，此时使用的锁对象是当前的字节码对象

案例：银行存款案例

# 4.死锁

互相等待产生死锁。死锁造成程序的假死状态，是很严重的问题，应该尽量避免死锁。

同步嵌套是造成死锁的必然原因，想要防止死锁，只要保证没有同步嵌套就可以了。

5.单例设计模式 -- 希望保证程序内指定的类的对象只有一个，整个程序范围内使用的该类的对象都是同一个。

将类的构造方法私有化。在类的内部保存静态的类的对象。提供getXxx方法，调用时返回类的内部中保存的该类的对象。

Person -- new Person() new Person() new Person()

Gov -- new Gov()

饿汉式：没有线程安全问题

懒汉式：有线程安全问题，需要用同步代码块进行安全保证，为了保证效率，还同步之前最好做一次额外的判断。

## 6.线程的调用过程

就绪 执行 阻塞 冻结

# 7.线程间的通信

案例：打印名称和性别

所谓的等待唤醒是在锁对象上进行的操作。因为锁对象可能是任意的对象，为了保证这个可能是任意的类的对象的锁对象具有这些方法，如下方法被定义在了Object类中。

wait -- 使当前线程进入等待状态（冻结） ，释放锁 释放cpu执行权 不再抢夺cpu 直到被其他线程唤醒

notify -- 唤醒一个线程，如果同时有多个线程处于wait状态，一旦notify随机唤醒一个线程（理论上是这样，实际上通常唤醒的都是第一个wait的线程），被唤醒后，线程仍然需要抢夺到锁后才能执行。

notifyall -- 唤醒所有线程

案例：生产者消费者问题

一个线程生产 另一个线程消费

多个线程生产 多个线程消费 如果使用notify可能产生多个线程都进入wait冻结状态的情况，造成程序类似死锁一样的效果

不要使用notify 应该使用notifyall方法

# 8.其他问题

## sleep和wait的区别：

sleep：睡眠

进入冻结状态 直到时间结束自动醒来 不再抢夺cpu 但是不释放锁 醒来后不意味着立即就能执行 仍然要抢到cpu才能执行

wait： 进入冻结状态

不再抢夺cpu 并且释放锁 直到被其他线程notify唤醒才能从等待状态中醒来 接着执行 并不意味着一定能立即执行 仍然需要抢夺到锁和cpu后才能执行

如何停止线程：

stop：此方法已经过时，具有固定的不安全性 ，所以不推荐使用。

当线程代码顺利执行完成后线程结束，所以可以通过标记控制线程的执行，来在需要的时候，通过修改标记来结束线程

这种方式停止线程，一旦线程处于冻结的状态，则即使改了标记，也没有机会执行判断，不能结束线程。此时可以调用interrupt() 中断线程，所谓的中断线程相当于将线程从冻结的状态强制唤醒过来。

## 守护线程：

setDaemon 将当指定线程设置位守护线程。线程其实分为前台线程 和 后台线程（守护线程）。

前台线程的特点是，不管其他人，只有自己结束时才结束。

后台线程（守护线程），除了自己结束时会结束以外，如果其他非守护都结束了，则守护线程无论是否结束都会立即结束掉。

加入线程：

join() 在当前线程执行过程中 临时加入一个线程 则当前线程就会冻结不再执行直到临时加入的线程执行完成 当期线程才会接着去执行

其他方法：

toString -- 获取线程信息的字符串

setPriority -- 设置线程的优先级，优先级越高执行到的机会就越大 0 - 5 - 10

yield -- 暂停当前线程 使其他线程有机会执行 此方法效果不明显 如果想实现线程间的通信 应该使用wait notify机制。