**进程：每个进程都有独立的代码和数据空间（进程上下文），进程间的切换会有较大的开销，一个进程包含1--n个线程。（进程是资源分配的最小单位）**

**线程：同一类线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器(PC)，线程切换开销小。（线程是cpu调度的最小单位）**

**线程和进程一样分为五个阶段：创建、就绪、运行、阻塞、终止。**

**一、扩展java.lang.Thread类**

继承Thread重写run()，然后start()

start()方法的调用后并不是立即执行多线程代码，而是使得该线程变为可运行态（Runnable），什么时候运行是由操作系统决定的。

# 二、实现java.lang.Runnable接口

实现Runnable接口，重写run()，然后通过Thread类的构造方法Thread(Runnable target) 构造出对象，然后调用Thread对象的start()方法来运行多线程代码。

**在java中，每次程序运行至少启动2个线程。一个是main线程，一个是垃圾收集线程。因为每当使用java命令执行一个类的时候，实际上都会启动一个ＪＶＭ，每一个ｊＶＭ就是在操作系统中启动了一个进程。**

**四、线程状态转换**



1、新建状态（New）：新创建了一个线程对象。

2、就绪状态（Runnable）：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态（Running）：就绪状态的线程获取了CPU，执行程序代码。

**4、阻塞状态（Blocked）：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：**

**（一）、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。(wait会释放持有的锁)**

**（二）、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。**

**（三）、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。（注意,sleep是不会释放持有的锁）**

5、死亡状态（Dead）：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

**五、线程调度**

线程的调度

1、调整线程优先级：Java线程有优先级，优先级高的线程会获得较多的运行机会。

Java线程的优先级用整数表示，取值范围是1~10，Thread类有以下三个静态常量：

static int MAX\_PRIORITY

          线程可以具有的最高优先级，取值为10。

static int MIN\_PRIORITY

          线程可以具有的最低优先级，取值为1。

static int NORM\_PRIORITY

          分配给线程的默认优先级，取值为5。

Thread类的setPriority()和getPriority()方法分别用来设置和获取线程的优先级。

每个线程都有默认的优先级。主线程的默认优先级为Thread.NORM\_PRIORITY。

线程的优先级有继承关系，比如A线程中创建了B线程，那么B将和A具有相同的优先级。

JVM提供了10个线程优先级，但与常见的操作系统都不能很好的映射。如果希望程序能移植到各个操作系统中，应该仅仅使用Thread类有以下三个静态常量作为优先级，这样能保证同样的优先级采用了同样的调度方式。

2、线程睡眠：Thread.sleep(long millis)方法，使线程转到阻塞状态。millis参数设定睡眠的时间，以毫秒为单位。当睡眠结束后，就转为就绪（Runnable）状态。sleep()平台移植性好。

3、线程等待：Object类中的wait()方法，导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 唤醒方法。这个两个唤醒方法也是Object类中的方法，行为等价于调用 wait(0) 一样。

4、线程让步：Thread.yield() 方法，暂停当前正在执行的线程对象，把执行机会让给相同或者更高优先级的线程。

5、线程加入：join()方法，等待其他线程终止。在当前线程中调用另一个线程的join()方法，则当前线程转入阻塞状态，直到另一个进程运行结束，当前线程再由阻塞转为就绪状态。

6、线程唤醒：Object类中的notify()方法，唤醒在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个线程。选择是任意性的，并在对实现做出决定时发生。线程通过调用其中一个 wait 方法，在对象的监视器上等待。 直到当前的线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。类似的方法还有一个notifyAll()，唤醒在此对象监视器上等待的所有线程。

 注意：Thread中suspend()和resume()两个方法在JDK1.5中已经废除，不再介绍。因为有死锁倾向。

**六、常用函数说明**

**①sleep(long millis): 在指定的毫秒数内让当前正在执行的线程休眠（暂停执行）  
  
②join():指等待t线程终止。**

**使用方式。**

join是Thread类的一个方法，启动线程后直接调用，即join()的作用是：“等待该线程终止”，这里需要理解的就是该线程是指的主线程等待子线程的终止。也就是在子线程调用了join()方法后面的代码，只有等到子线程结束了才能执行。

**③yield():暂停当前正在执行的线程对象，并执行其他线程。**

        Thread.yield()方法作用是：暂停当前正在执行的线程对象，并执行其他线程。

**yield()应该做的是让当前运行线程回到可运行状态，以允许具有相同优先级的其他线程获得运行机会。**因此，使用yield()的目的是让相同优先级的线程之间能适当的轮转执行。但是，实际中无法保证yield()达到让步目的，因为让步的线程还有可能被线程调度程序再次选中。

**sleep()和yield()的区别**  
        sleep()和yield()的区别):sleep()使当前线程进入停滞状态，所以执行sleep()的线程在指定的时间内肯定不会被执行；yield()只是使当前线程重新回到可执行状态，所以执行yield()的线程有可能在进入到可执行状态后马上又被执行。  
        sleep 方法使当前运行中的线程睡眼一段时间，进入不可运行状态，这段时间的长短是由程序设定的，yield 方法使当前线程让出 CPU 占有权，但让出的时间是不可设定的。实际上，yield()方法对应了如下操作：先检测当前是否有相同优先级的线程处于同可运行状态，如有，则把 CPU  的占有权交给此线程，否则，继续运行原来的线程。所以yield()方法称为“退让”，它把运行机会让给了同等优先级的其他线程  
       另外，sleep 方法允许较低优先级的线程获得运行机会，但 yield()  方法执行时，当前线程仍处在可运行状态，所以，不可能让出较低优先级的线程些时获得 CPU 占有权。在一个运行系统中，如果较高优先级的线程没有调用 sleep 方法，又没有受到 I\O 阻塞，那么，较低优先级线程只能等待所有较高优先级的线程运行结束，才有机会运行。   
  
**④setPriority(): 更改线程的优先级。**

　　　　MIN\_PRIORITY = 1  
  　　   NORM\_PRIORITY = 5  
           MAX\_PRIORITY = 10

用法：

Thread4 t1 = new Thread4("t1");

Thread4 t2 = new Thread4("t2");

t1.setPriority(Thread.MAX\_PRIORITY);

t2.setPriority(Thread.MIN\_PRIORITY);

**⑤interrupt():不要以为它是中断某个线程！它只是线线程发送一个中断信号，让线程在无限等待时（如死锁时）能抛出抛出，从而结束线程，但是如果你吃掉了这个异常，那么这个线程还是不会中断的！**

⑥wait()

Obj.wait()，与Obj.notify()必须要与synchronized(Obj)一起使用，也就是wait,与notify是针对已经获取了Obj锁进行操作，从语法角度来说就是Obj.wait(),Obj.notify必须在synchronized(Obj){...}语句块内。从功能上来说wait就是说线程在获取对象锁后，主动释放对象锁，同时本线程休眠。直到有其它线程调用对象的notify()唤醒该线程，才能继续获取对象锁，并继续执行。相应的notify()就是对对象锁的唤醒操作。但有一点需要注意的是notify()调用后，并不是马上就释放对象锁的，而是在相应的synchronized(){}语句块执行结束，自动释放锁后，JVM会在wait()对象锁的线程中随机选取一线程，赋予其对象锁，唤醒线程，继续执行。这样就提供了在线程间同步、唤醒的操作。Thread.sleep()与Object.wait()二者都可以暂停当前线程，释放CPU控制权，主要的区别在于Object.wait()在释放CPU同时，释放了对象锁的控制。

wait和sleep区别

共同点：

1. 他们都是在多线程的环境下，都可以在程序的调用处阻塞指定的毫秒数，并返回。

2. wait()和sleep()都可以通过interrupt()方法 打断线程的暂停状态 ，从而使线程立刻抛出InterruptedException。

如果线程A希望立即结束线程B，则可以对线程B对应的Thread实例调用interrupt方法。如果此刻线程B正在wait/sleep /join，则线程B会立刻抛出InterruptedException，在catch() {} 中直接return即可安全地结束线程。

需要注意的是，InterruptedException是线程自己从内部抛出的，并不是interrupt()方法抛出的。对某一线程调用 interrupt()时，如果该线程正在执行普通的代码，那么该线程根本就不会抛出InterruptedException。但是，一旦该线程进入到 wait()/sleep()/join()后，就会立刻抛出InterruptedException 。

不同点：

1. Thread类的方法：sleep(),yield()等

Object的方法：wait()和notify()等

2. 每个对象都有一个锁来控制同步访问。Synchronized关键字可以和对象的锁交互，来实现线程的同步。

sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法。

3. 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

4. sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常

所以sleep()和wait()方法的最大区别是：

　　　　sleep()睡眠时，保持对象锁，仍然占有该锁；

　　　　而wait()睡眠时，释放对象锁。

　　但是wait()和sleep()都可以通过interrupt()方法打断线程的暂停状态，从而使线程立刻抛出InterruptedException（但不建议使用该方法）。

sleep（）方法

sleep()使当前线程进入停滞状态（阻塞当前线程），让出CUP的使用、目的是不让当前线程独自霸占该进程所获的CPU资源，以留一定时间给其他线程执行的机会;

　　 sleep()是Thread类的Static(静态)的方法；因此他不能改变对象的机锁，所以当在一个Synchronized块中调用Sleep()方法是，线程虽然休眠了，但是对象的机锁并木有被释放，其他线程无法访问这个对象（即使睡着也持有对象锁）。

　　在sleep()休眠时间期满后，该线程不一定会立即执行，这是因为其它线程可能正在运行而且没有被调度为放弃执行，除非此线程具有更高的优先级。

wait（）方法

wait()方法是Object类里的方法；当一个线程执行到wait()方法时，它就进入到一个和该对象相关的等待池中，同时失去（释放）了对象的机锁（暂时失去机锁，wait(long timeout)超时时间到后还需要返还对象锁）；其他线程可以访问；

　　wait()使用notify或者notifyAlll或者指定睡眠时间来唤醒当前等待池中的线程。

wiat()必须放在synchronized block中，否则会在program runtime时扔出”java.lang.IllegalMonitorStateException“异常。

**线程类的一些常用方法：   
  
　　sleep(): 强迫一个线程睡眠Ｎ毫秒。   
　　isAlive(): 判断一个线程是否存活。   
　　join(): 等待线程终止。   
　　activeCount(): 程序中活跃的线程数。   
　　enumerate(): 枚举程序中的线程。   
    currentThread(): 得到当前线程。   
　　isDaemon(): 一个线程是否为守护线程。   
　　setDaemon(): 设置一个线程为守护线程。(用户线程和守护线程的区别在于，是否等待主线程依赖于主线程结束而结束)   
　　setName(): 为线程设置一个名称。   
　　wait(): 强迫一个线程等待。   
　　notify(): 通知一个线程继续运行。   
　　setPriority(): 设置一个线程的优先级。**

**八、线程同步**

1、synchronized关键字的作用域有二种：   
1）是某个对象实例内，synchronized aMethod(){}可以防止多个线程同时访问这个对象的synchronized方法（如果一个对象有多个synchronized方法，只要一个线程访问了其中的一个synchronized方法，其它线程不能同时访问这个对象中任何一个synchronized方法）。这时，不同的对象实例的synchronized方法是不相干扰的。也就是说，其它线程照样可以同时访问相同类的另一个对象实例中的synchronized方法；   
2）是某个类的范围，synchronized static aStaticMethod{}防止多个线程同时访问这个类中的synchronized static 方法。它可以对类的所有对象实例起作用。   
  
2、除了方法前用synchronized关键字，synchronized关键字还可以用于方法中的某个区块中，表示只对这个区块的资源实行互斥访问。用法是: synchronized(this){/\*区块\*/}，它的作用域是当前对象；   
  
3、synchronized关键字是不能继承的，也就是说，基类的方法synchronized f(){} 在继承类中并不自动是synchronized f(){}，而是变成了f(){}。继承类需要你显式的指定它的某个方法为synchronized方法；

总的说来，synchronized关键字可以作为函数的修饰符，也可作为函数内的语句，也就是平时说的同步方法和同步语句块。如果再细的分类，synchronized可作用于instance变量、object reference（对象引用）、static函数和class literals(类名称字面常量)身上。

在进一步阐述之前，我们需要明确几点：

A．无论synchronized关键字加在方法上还是对象上，它取得的锁都是对象，而不是把一段代码或函数当作锁――而且同步方法很可能还会被其他线程的对象访问。

B．每个对象只有一个锁（lock）与之相关联。

C．实现同步是要很大的系统开销作为代价的，甚至可能造成死锁，所以尽量避免无谓的同步控制。

接着来讨论synchronized用到不同地方对代码产生的影响：

假设P1、P2是同一个类的不同对象，这个类中定义了以下几种情况的同步块或同步方法，P1、P2就都可以调用它们。

1．  把synchronized当作函数修饰符时，示例代码如下：

Public synchronized void methodAAA()

{

//….

}

这也就是同步方法，那这时synchronized锁定的是哪个对象呢？它锁定的是调用这个同步方法对象。也就是说，当一个对象P1在不同的线程中执行这个同步方法时，它们之间会形成互斥，达到同步的效果。但是这个对象所属的Class所产生的另一对象P2却可以任意调用这个被加了synchronized关键字的方法。

上边的示例代码等同于如下代码：

public void methodAAA()

{

synchronized (this)      //  (1)

{

       //…..

}

}

 (1)处的this指的是什么呢？它指的就是调用这个方法的对象，如P1。可见同步方法实质是将synchronized作用于object reference。――那个拿到了P1对象锁的线程，才可以调用P1的同步方法，而对P2而言，P1这个锁与它毫不相干，程序也可能在这种情形下摆脱同步机制的控制，造成数据混乱：（

2．同步块，示例代码如下：

            public void method3(SomeObject so)

              {

                     synchronized(so)

{

       //…..

}

}

这时，锁就是so这个对象，谁拿到这个锁谁就可以运行它所控制的那段代码。当有一个明确的对象作为锁时，就可以这样写程序，但当没有明确的对象作为锁，只是想让一段代码同步时，可以创建一个特殊的instance变量（它得是一个对象）来充当锁：

class Foo implements Runnable

{

       private byte[] lock = new byte[0];  // 特殊的instance变量

    Public void methodA()

{

       synchronized(lock) { //… }

}

//…..

}

注：零长度的byte数组对象创建起来将比任何对象都经济――查看编译后的字节码：生成零长度的byte[]对象只需3条操作码，而Object lock = new Object()则需要7行操作码。

3．将synchronized作用于static 函数，示例代码如下：

      Class Foo

{

public synchronized static void methodAAA()   // 同步的static 函数

{

//….

}

public void methodBBB()

{

       synchronized(Foo.class)   //  class literal(类名称字面常量)

}

       }

   代码中的methodBBB()方法是把class literal作为锁的情况，它和同步的static函数产生的效果是一样的，取得的锁很特别，是当前调用这个方法的对象所属的类（Class，而不再是由这个Class产生的某个具体对象了）。

记得在《Effective Java》一书中看到过将 Foo.class和 P1.getClass()用于作同步锁还不一样，不能用P1.getClass()来达到锁这个Class的目的。P1指的是由Foo类产生的对象。

可以推断：如果一个类中定义了一个synchronized的static函数A，也定义了一个synchronized 的instance函数B，那么这个类的同一对象Obj在多线程中分别访问A和B两个方法时，不会构成同步，因为它们的锁都不一样。A方法的锁是Obj这个对象，而B的锁是Obj所属的那个Class。

1、线程同步的目的是为了保护多个线程反问一个资源时对资源的破坏。

2、线程同步方法是通过锁来实现，每个对象都有切仅有一个锁，这个锁与一个特定的对象关联，线程一旦获取了对象锁，其他访问该对象的线程就无法再访问该对象的其他非同步方法。

3、对于静态同步方法，锁是针对这个类的，锁对象是该类的Class对象。静态和非静态方法的锁互不干预。一个线程获得锁，当在一个同步方法中访问另外对象上的同步方法时，会获取这两个对象锁。

4、对于同步，要时刻清醒在哪个对象上同步，这是关键。

5、编写线程安全的类，需要时刻注意对多个线程竞争访问资源的逻辑和安全做出正确的判断，对“原子”操作做出分析，并保证原子操作期间别的线程无法访问竞争资源。

6、当多个线程等待一个对象锁时，没有获取到锁的线程将发生阻塞。

7、死锁是线程间相互等待锁锁造成的，在实际中发生的概率非常的小。真让你写个死锁程序，不一定好使，呵呵。但是，一旦程序发生死锁，程序将死掉。

# 九、线程数据传递

9.1、通过构造方法传递数据

9.2、通过变量和方法传递数据

9.3、通过回调函数传递数据