1. 线程
2. 多线程原理

程序启动进入main函数，JVM启动了一个进程，主线程main在main()被调用时创建，主线程运行在自己的栈空间中。之后一个新的线程启动，新的线程栈空间被创建。每个线程有自己的栈空间，当执行线程的任务结束时，线程自动在栈空间中释放了；当所有线程被释放时，进程就结束了。

1. Thread类的构造方法
2. Thread类直接构造一个新的线程对象

public Thread() :分配一个新的线程对象。

public Thread(String name) :分配一个指定名字的新的线程对象。

1. 用Runnable接口的实现类来改造新的线程对象

2.1构造方法

public Thread(Runnable target) :分配一个带有指定目标新的线程对象。

public Thread(Runnable target,String name) :分配一个带有指定目标新的线程对象并指定名字。

2.2步骤

（1）定义Runnable接口的实现类，并重写该接口的run()方法，该run()方法的方法体同样是该线程的线程执行体。

（2）创建Runnable实现类的实例，并以此实例作为Thread的target来创建Thread对象，该Thread对象才是真正

的线程对象。

（3）调用线程对象的start()方法来启动线程。

2.3 Runnable接口的优点

（1）用Thread类直接构造新线程，有单继承的局限性，而Runnable接口的实现类可以实现多个接口，也可以再继承一个类。

（2）多个线程可以共享同一个资源，传入同一个Runnable的实现类。

（3）增强了程序的扩展性，降低了程序的耦合性。设置线程任务和开启新线程进行了分离。Runnable接口的实现类中重写了run()方法，用来设置线程任务；在创建新线程时，创建Thread类对象，调用start()方法，用来开启新线程，二者是分离的。

（4）线程池只能放入实现Runable或Callable类线程，不能直接放入继承Thread的类。

1. Thread类的常用方法

public String getName() :获取当前线程名称。

public void start() :导致此线程开始执行; Java虚拟机调用此线程的run方法。

public void run() :此线程要执行的任务在此处定义代码。

public static void sleep(long millis) :使当前正在执行的线程以指定的毫秒数暂停（暂时停止执行）。

public static Thread currentThread() :返回对当前正在执行的线程对象的引用。

1. 匿名内部类创建线程

使用匿名内部类的方式，可以快速地创建新线程。

1. 创建Thread类的匿名内部类

//Thread()  
new Thread(){  
 @Override  
 public void run() {  
 for(int i = 0 ; i < 20; ++i){  
 System.*out*.println(Thread.*currentThread*().getName() + i);  
 }  
 }  
}.start();

1. 创建Runnable接口的匿名内部类

//Runnable  
new Thread(new Runnable() {  
 @Override  
 public void run() {  
 for(int i = 0; i <10; ++i){  
 System.*out*.println(i);  
 }  
 }  
}).start();

1. 线程安全
2. 概述

线程安全问题一般由全局变量和静态变量引起，若每个线程中对全局变量、静态变量只有读操作，而无写

操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作，一般都需要考虑线程同步，

否则的话就可能影响线程安全。

1. 线程同步

要解决多线程并发访问同一个资源的安全性问题，就要用到同步机制（synchronized）。

1. 同步代码块

将synchronized关键字用于某个块代码中，表示只对这个块代码进行互斥访问。

1. 格式：

synchronized(同步锁){

     需要同步操作的代码

}

1. 同步锁

同步锁的类型可以是任意类型，谁拿到了这个锁，谁就能够进入代码块执行代码，而其他没有拿到锁的线程只能等待（BLOCKED）。多个线程使用的是同一把锁。

1. 同步方法

用synchronized关键字修饰的方法称为同步方法。

1. 格式

public synchronized void method(){

   可能会产生线程安全问题的代码

}

1. 同步锁

对于非static方法，同步锁是this，即调用该方法的对象。

对于static方法，同步锁是调用该方法所在类的字节码对象。（类名.class）。

1. Lock锁

更加广泛的锁操作，将加锁和释放锁都变成了方法：

public void lock() :加同步锁。

public void unlock() :释放同步锁。

在需要加锁的代码前加锁，在代码后释放锁。

1. 线程状态
2. 六种状态

线程状态 导致状态发生条件

NEW(新建) 线程刚被创建，但是并未启动。还没调用start方法。

Runnable(可

运行)

线程可以在java虚拟机中运行的状态，可能正在运行自己代码，也可能没有，这取决于操

作系统处理器。

Blocked(锁阻

塞)

当一个线程试图获取一个对象锁，而该对象锁被其他的线程持有，则该线程进入Blocked状

态；当该线程持有锁时，该线程将变成Runnable状态。

Waiting(无限

等待)

一个线程在等待另一个线程执行一个（唤醒）动作时，该线程进入Waiting状态。进入这个

状态后是不能自动唤醒的，必须等待另一个线程调用notify或者notifyAll方法才能够唤醒。

Timed

Waiting(计时

等待)

同waiting状态，有几个方法有超时参数，调用他们将进入Timed Waiting状态。这一状态

将一直保持到超时期满或者接收到唤醒通知。带有超时参数的常用方法有Thread.sleep 、

Object.wait。

Teminated(被

终止)

因为run方法正常退出而死亡，或者因为没有捕获的异常终止了run方法而死亡。

1. Time Waiting

计时等待，可以通过Thread.sleep，Object.wait进入线程的等待，并自动唤醒或者受到唤醒通知。

Sleep与锁无关，线程等待到期自动唤醒，返回到Runnable状态。注意，sleep()中指定的时间是线程等待的最短时间，并不能保证该时间到期就一定唤醒。

1. BLOCKED

没有争取到锁对象的线程进入阻塞状态，等待锁对象。

1. Waiting

无限等待状态，调用了Object.wait方法只能等待另一个线程调用该对象的Object.notify()方法或者Object.notifyAll()方法唤醒。被唤醒后的线程如果没有锁对象，就会处于BLOCKED状态，有了锁对象，才是Runnable状态。