一、Thread和Runnable；

run 和start

二、获取Thread的信息

id、name、优先级、状态

三。线程的中断

interrupt();

isInterrupt();

用InterruptedException来控制中断

四、线程的sleep和wake

TimeUnit.SECONDS.sleep(秒)；

join()等待线程的终止；

五、守护线程

提供服务的，所有用户线程退出，它也会退出；

先设置守护线程setDaemon（true）；

六、运行时异常捕捉

定义一个异常类实现UncaughtExceptionHandler接口；

Thread 中的setUncaughtExceptionHandler（）；

七、线程的变量

如果是实现Runnable接口的类，当做参数传入Thread中，启动的多个线程共享变量；

为了隔离，使用ThreadLocal

八、线程组

ThreadGroup 声明一个标识的组

九、工厂模式

总体来说：

1、Thread是类，Runable是接口，继承Thread或实现Runable接口都可以创建一个线程；

2、Thread除了run()方法还有start()方法，run()方法里面是线程要做的工作！start()是线程交互执行的启动方法！（因为Thread类中start()方法前面的修饰关键字是native，native表示该方法不是调用java语言方法，而是调用操作系统底层的方法！）

3、单纯执行Thread无法对同一资源进行多线程并发操作，使用Runable接口就可以！

如下例：

public class Test {

    public static void main(String[] args) {

        /\*\*

        Thread thread1 = new MyThread("线程1");

        Thread thread2 = new MyThread("线程2");

        Thread thread3 = new MyThread("线程3");

        Thread thread4 = new MyThread("线程4");

        thread1.start();

        thread2.start();

        thread3.start();

        thread3.run();

        //System.out.println("++++++");

        System.exit(0);

        \*/

        Runable runable1 = new MyRunable()；

        new Thread(runable1 ).start();

        new Thread(runable1 ).start();

    }

}

class MyThread extends Thread {

    private String name;

    private int ticket = 1000;

    public MyThread(String name) {

        this.name = name;

    }

    public void run() {

        while(ticket>0) {

            System.out.println(name + "还剩：" + (ticket--));

        }

    }

}

class MyRunable implements Runnable {

    private String name;

    private int ticket = 1000;

    @Override

    public void run() {

       buy();

    }

   public void buy() {

          while(ticket>0) {

              System.out.println(name + "还剩：" + (ticket--));

            }

   }

}

MyThread是继承Thread实现的一个线程，MyRunable是实现Runable接口的线程

第一种情况：thread1.run();

                     thread2.run();

    这时候run()只是个线程类的普通函数，执行完线程1的run方法后，再执行线程2的run方法；结果会是线程1的票从1000到1，线程2的票从1000到1

第二种情况：

                     thread1.run();

                     thread1.start();

   这时候thread1对象只执行一个方法，run在前面执行run，start在前面执行start();

第三种情况：

                     thread1.start();

                     thread2.start();

   这时候线程1和线程2交替执行；但是由于thread1和thread2是不同的对象，都执行自己的run方法，各自卖自己的1000张票，那么总共就买了2000张票；

第四种情况：thread1.start();

                     thread1.start();

   同一个线程对象，启动两次start(),只执行一次，且程序会报错！

第五种情况：Runnable runable1 = new MyRunable()；

                     new Thread(runable1 ).start();

                     new Thread(runable1 ).start();

 Runnable自身只有run()方法，没有start()方法，所以它自己无法作为一个线程启动；但是Thread里面有一个构造函数的参数就是Runnable，所以可以用Thread类进行包装，上述，runable1 是一个对象，但是包装了两次，所以启动了两个线程，这两个线程可以共享runable1 里面的资源，所以，输出结果为

两个线程交替输出：1000到1，但是次序可能会出现混乱；因为当ticket=1000时，线程1执行完 System.out.println(name + "还剩：" + (ticket--));但是还未执行ticket--就交给线程2执行，此时由于未进行ticket--操作，故ticke还是为1000，线程执行完System.out.println(name + "还剩：" + (ticket--));此时会出现线程1和线程2同时输出1000；

这就是多线程的同步问题！

多线程同步，就是让多个线程在访问同一个资源的时候（前提是这个资源必须是要求互斥访问的）只能互斥访问！

这里有个问题，资源不仅仅只是cpu；如果一个cpu，多个线程那么对cpu的使用是分时间片轮流进行的，但是对其他资源，如内存等等并不是按cpu分片进行访问的；比如线程1占用cpu的同时访问一块内存，如果线程2此时没有挂起或者阻塞等，他也是可以访问这块内存的！对于多cpu的情况，就更加复杂了！所以，资源的互斥操作是非常重要的！

如对方法的访问，方法及其参数都是放在内存中的，不同线程对同一方法的访问就会引起混乱，所有必须对这一方法进行同步；

方法的同步有两种方式，synchronized关键字和synchronized（lock）{。。。}块；其原理都是给方法一个唯一的锁！任何线程访问这个方法都得进过这把锁！

    用关键字synchronized，上述方法可以改写为

 publicsynchronized void buy() {

          while(ticket>0) {

              System.out.println(name + "还剩：" + (ticket--));

            }

   }

此时输出就不会出现混乱了，但是如果Runnable对象不是一个而是两个，如下代码：

                        new Thread(runable1 ).start();

                       new Thread(runable2).start();

那么这个synchronized 关键字就没有作用，这是因为synchronized 关键字针对的是this，指的是当前对象给一个锁，既然两个对象，就是两个锁，所以这个锁就没有意义了！

这时候可以用synchronized（lock）{。。。}块，为了保证这个锁唯一，可以改写代码:

class MyRunable implements Runnable {

    private static Object lock = new Object ();

    private int ticket = 1000;

    @Override

    public void run() {

       buy();

    }

 synchronized(lock ) {

           public void buy() {

                      while(ticket>0) {

                              System.out.println(name + "还剩：" + (ticket--));

                        }

           }

    }

}

上述代码中，因为静态变量是属于类的，而不属于对象的，所以，无论创建多少个MyRunable 对象，其lock锁始终是一个！