Java中创建线程对象有两种方式

1.自定义一个线程类，继承Thread类，重写父类的run方法，完成自定义线程的具体的业务逻辑。

使用：实例化自定义线程类的对象，该对象就是一个线程对象，然后启动线程即可。

```
//自定义线程类
public class MyThread extends Thread{
    //自定义线程的具体的业务逻辑
    public void run(){
        dosomething...
    }
}

//使用
MyThread myThread = MyThread();
myThread.start();
```

2.自定义一个类，实现Runnable接口，实现接口的run方法，完成自定义线程的具体业务逻辑。

使用：实例化实现类的对象，但是它并不是一个线程对象，它只是实现了线程要执行的业务逻辑，还需要实例化一个线程对象（Thread），将实现类对象传给线程对象，就会覆盖线程对象继承自父类的run方法，完成自定义的业务逻辑。

```
//自定义Runnable实现类
public class MyRunnable implements Runnable{
    //自定义线程要执行的具体的业务逻辑
    public void run(){
        dosomething...
    }
}

//使用
MyRunnable myRunnable = new MyRunnable();
Thread thread = new Thread(myRunnable);
thread.start();
```

注意

启动线程需要调用线程对象的start方法，不能调用run方法，调用run方法相当于普通的方法调用，不是多线程，应该调用start方法来启动该线程，然后该线程就进入了就绪状态，等待系统为其分配CPU资源，一旦在时间段内获取到CPU资源，开始执行线程的业务逻辑，自动调用run。

线程的状态

1.创建状态：一个线程对象被实例化之后。

2.就绪状态：创建好的线程对象调用了start方法启动之后。

3.运行状态：当前线程对象获取了CPU资源，可以执行其业务逻辑代码（run方法的代码）。

4.阻塞状态：运行状态的线程进行了休眠或其他操作导致线程暂停执行。

5.死亡状态：线程执行完毕或者因为异常退出。

线程调度

- 线程休眠

休眠指让当前线程暂停执行一段时间，从运行状态进入阻塞状态，将CPU资源让给其他线程的一种调度方法，通过sleep方法来实现，sleep(long millis)是java.lang.Thread类中定义的方法，使用时需要指定当前线程休眠的时间，单位为毫秒（1000毫秒=1秒）。

```
public class Test2 {
    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        try {
            myThread.sleep(3000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        myThread.start();
    }
}

class MyThread extends Thread{
    @Override
    public void run() {
        // TODO Auto-generated method stub
        for (int i = 0; i < 10; i++) {
            if(i == 5) {
                try {
                    sleep(3000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            System.out.println(i);
        }
    }
}
```

mian线程进行休眠操作

```
public class Test3 {
```

```
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if(i == 5) {
                try {
                    Thread.currentThread().sleep(3000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            System.out.println(i);
        }
    }
}
```

- 线程合并

合并的意思是将指定的某个线程合并到当前线程中，将原本两个交替执行的线程改为顺序执行，即一个线程执行完毕之后再来执行第二个线程，通过调用线程对象的join方法来实现合并。

具体如何来实现合并？谁为主谁为从？假设有两个线程：线程甲，线程乙。

线程甲在执行到某个时间点的时候调用了线程乙的join方法，则表示从当前时间点开始CPU资源被线程乙独占，线程甲进行阻塞状态，直到线程乙执行完毕，线程甲重新进入就绪状态，等待获取CPU资源进入运行状态继续执行。

```
package com.southwind.thread;

public class Test4 {
    public static void main(String[] args) {
        JoinRunnable joinRunnable = new JoinRunnable();
        Thread thread = new Thread(joinRunnable);
        thread.start();
        for (int i = 0; i < 100; i++) {
            if(i == 10) {
                try {
                    //thread.join(3000);
                    thread.join();
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            System.out.println(i+"Test++++++++++++++++++");
        }
    }
}

class JoinRunnable implements Runnable{
```

```
    @Override
    public void run() {
        // TODO Auto-generated method stub
        for (int i = 0; i < 10; i++) {
            try {
                Thread.currentThread().sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println(i+"------------JoinRunnable");
        }
    }

}
```

同样是完成线程合并的操作，join()和join(long millis)是有区别的，join()表示被调用线程执行完成之后才能释放CPU资源，让其他线程来执行，join(long millis)则表示被调用的线程执行millis毫秒之后，无论其是否执行完毕，其他线程都可以和它来争夺CPU资源。

- 线程礼让

线程礼让是指在某个特定的时间点，让当前线程暂停抢占CPU资源的行为，即从运行状态或就绪状态来到阻塞状态，从而将CPU资源让给其他线程来使用。假如有线程甲和线程乙在交替执行，某个时间点线程甲作出了礼让，所以在这个时间点线程乙就拥有了CPU资源，执行其业务逻辑，但不是说线程甲会一直暂停争夺，线程甲只是在特定的时间节点进行礼让，一旦过了这个时间节点，线程甲在此进入就绪状态，和线程乙来争夺CPU资源。

通过调用线程对象的yield方法来完成礼让。

```
package com.southwind.thread;

public class Test5 {
    public static void main(String[] args) {
        YieldThread1 y1 = new YieldThread1();
        y1.setName("YieldThread1");
        YieldThread2 y2 = new YieldThread2();
        y2.setName("YieldThread2");
        y1.start();
        y2.start();
    }
}

class YieldThread1 extends Thread{
    @Override
    public void run() {
        // TODO Auto-generated method stub
        for (int i = 0; i < 10; i++) {
            if(i == 5) {
                yield();
```

```
            }
            System.out.println(getName()+"------"+i);
        }
    }
}

class YieldThread2 extends Thread{
    @Override
    public void run() {
        // TODO Auto-generated method stub
        for (int i = 0; i < 30; i++) {
            System.out.println(getName()+"------"+i);
        }
    }
}
```

- 线程中断

Java中实现线程中断机制有如下几个方法：

public void stop()

public void interrupt()

public boolean isInterrupt()

stop()方法在新版本的JDK已经不推荐使用了，interrupt是一个实例方法，当一个线程对象调用该方法时，表示中断当前线程对象。

每一个线程都有一个标志位来标识当前线程对象是否为中断状态，isInterrupt()方法就是用来获取当前线程对象的标志位的，true表示清除了标志位，当前线程对象已经中断，false表示没有清除标志位，当前线程对象没有中断。

```
package com.southwind.thread;

public class Test6 {
    public static void main(String[] args) {
        Thread thread = new Thread();
        System.out.println(thread.getState());
        thread.start();
        System.out.println(thread.isInterrupted());
        thread.interrupt();
        System.out.println(thread.isInterrupted());
    }
}
```