**Java多线程 概述**

**一.线程的生命周期及五种基本状态**

关于Java中线程的生命周期，首先看一下下面这张较为经典的图：



上图中基本上囊括了Java中多线程各重要知识点。掌握了上图中的各知识点，Java中的多线程也就基本上掌握了。主要包括：

**Java线程具有五中基本状态：**

**新建状态（New）：**当线程对象对创建后，即进入了新建状态，如：Thread t = new MyThread();

**就绪状态（Runnable）：**当调用线程对象的start()方法（t.start();），线程即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时等待CPU调度执行，并不是说执行了t.start()此线程立即就会执行；

**运行状态（Running）：**当CPU开始调度处于就绪状态的线程时，此时线程才得以真正执行，即进入到运行状态。注：就绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，首先必须处于就绪状态中；

**阻塞状态（Blocked）：**处于运行状态中的线程由于某种原因，暂时放弃对CPU的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有机会再次被CPU调用以进入到运行状态。根据阻塞产生的原因不同，阻塞状态又可以分为三种：

1.等待阻塞：运行状态中的线程执行wait()方法，使本线程进入到等待阻塞状态；

2.同步阻塞 -- 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，它会进入同步阻塞状态；

3.其他阻塞 -- 通过调用线程的sleep()或join()或发出了I/O请求时，线程会进入到阻塞状态。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。

**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。

**二. Java多线程的创建及启动**

Java中线程的创建常见有如三种基本形式

**1.继承Thread类，重写该类的run()方法。**

[复制代码](javascript:void(0);)

1 class MyThread extends Thread {

2

3 private int i = 0;

4

5 @Override

6 public void run() {

7 for (i = 0; i < 100; i++) {

8 System.out.println(Thread.currentThread().getName() + " " + i);

9 }

10 }

11 }

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 public class ThreadTest {

2

3 public static void main(String[] args) {

4 for (int i = 0; i < 100; i++) {

5 System.out.println(Thread.currentThread().getName() + " " + i);

6 if (i == 30) {

7 Thread myThread1 = new MyThread(); // 创建一个新的线程 myThread1 此线程进入新建状态

8 Thread myThread2 = new MyThread(); // 创建一个新的线程 myThread2 此线程进入新建状态

9 myThread1.start(); // 调用start()方法使得线程进入就绪状态

10 myThread2.start(); // 调用start()方法使得线程进入就绪状态

11 }

12 }

13 }

14 }

[复制代码](javascript:void(0);)

如上所示，继承Thread类，通过重写run()方法定义了一个新的线程类MyThread，其中run()方法的方法体代表了线程需要完成的任务，称之为线程执行体。当创建此线程类对象时一个新的线程得以创建，并进入到线程新建状态。通过调用线程对象引用的start()方法，使得该线程进入到就绪状态，此时此线程并不一定会马上得以执行，这取决于CPU调度时机。

**2.实现Runnable接口，并重写该接口的run()方法，该run()方法同样是线程执行体，创建Runnable实现类的实例，并以此实例作为Thread类的target来创建Thread对象，该Thread对象才是真正的线程对象。**

[复制代码](javascript:void(0);)

1 class MyRunnable implements Runnable {

2 private int i = 0;

3

4 @Override

5 public void run() {

6 for (i = 0; i < 100; i++) {

7 System.out.println(Thread.currentThread().getName() + " " + i);

8 }

9 }

10 }

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 public class ThreadTest {

2

3 public static void main(String[] args) {

4 for (int i = 0; i < 100; i++) {

5 System.out.println(Thread.currentThread().getName() + " " + i);

6 if (i == 30) {

7 Runnable myRunnable = new MyRunnable(); // 创建一个Runnable实现类的对象

8 Thread thread1 = new Thread(myRunnable); // 将myRunnable作为Thread target创建新的线程

9 Thread thread2 = new Thread(myRunnable);

10 thread1.start(); // 调用start()方法使得线程进入就绪状态

11 thread2.start();

12 }

13 }

14 }

15 }

[复制代码](javascript:void(0);)

相信以上两种创建新线程的方式大家都很熟悉了，那么Thread和Runnable之间到底是什么关系呢？我们首先来看一下下面这个例子。

[复制代码](javascript:void(0);)

1 public class ThreadTest {

2

3 public static void main(String[] args) {

4 for (int i = 0; i < 100; i++) {

5 System.out.println(Thread.currentThread().getName() + " " + i);

6 if (i == 30) {

7 Runnable myRunnable = new MyRunnable();

8 Thread thread = new MyThread(myRunnable);

9 thread.start();

10 }

11 }

12 }

13 }

14

15 class MyRunnable implements Runnable {

16 private int i = 0;

17

18 @Override

19 public void run() {

20 System.out.println("in MyRunnable run");

21 for (i = 0; i < 100; i++) {

22 System.out.println(Thread.currentThread().getName() + " " + i);

23 }

24 }

25 }

26

27 class MyThread extends Thread {

28

29 private int i = 0;

30

31 public MyThread(Runnable runnable){

32 super(runnable);

33 }

34

35 @Override

36 public void run() {

37 System.out.println("in MyThread run");

38 for (i = 0; i < 100; i++) {

39 System.out.println(Thread.currentThread().getName() + " " + i);

40 }

41 }

42 }

[复制代码](javascript:void(0);)

同样的，与实现Runnable接口创建线程方式相似，不同的地方在于

1 Thread thread = new MyThread(myRunnable);

那么这种方式可以顺利创建出一个新的线程么？答案是肯定的。至于此时的线程执行体到底是MyRunnable接口中的run()方法还是MyThread类中的run()方法呢？通过输出我们知道线程执行体是MyThread类中的run()方法。其实原因很简单，因为Thread类本身也是实现了Runnable接口，而run()方法最先是在Runnable接口中定义的方法。

1 public interface Runnable {

2

3 public abstract void run();

4

5 }

我们看一下Thread类中对Runnable接口中run()方法的实现：

　　@Override

public void run() {

if (target != null) {

target.run();

}

}

也就是说，当执行到Thread类中的run()方法时，会首先判断target是否存在，存在则执行target中的run()方法，也就是实现了Runnable接口并重写了run()方法的类中的run()方法。但是上述给到的列子中，由于多态的存在，根本就没有执行到Thread类中的run()方法，而是直接先执行了运行时类型即MyThread类中的run()方法。

**3.使用Callable和Future接口创建线程。具体是创建Callable接口的实现类，并实现clall()方法。并使用FutureTask类来包装Callable实现类的对象，且以此FutureTask对象作为Thread对象的target来创建线程。**

 看着好像有点复杂，直接来看一个例子就清晰了。

[复制代码](javascript:void(0);)

1 public class ThreadTest {

2

3 public static void main(String[] args) {

4

5 Callable<Integer> myCallable = new MyCallable(); // 创建MyCallable对象

6 FutureTask<Integer> ft = new FutureTask<Integer>(myCallable); //使用FutureTask来包装MyCallable对象

7

8 for (int i = 0; i < 100; i++) {

9 System.out.println(Thread.currentThread().getName() + " " + i);

10 if (i == 30) {

11 Thread thread = new Thread(ft); //FutureTask对象作为Thread对象的target创建新的线程

12 thread.start(); //线程进入到就绪状态

13 }

14 }

15

16 System.out.println("主线程for循环执行完毕..");

17

18 try {

19 int sum = ft.get(); //取得新创建的新线程中的call()方法返回的结果

20 System.out.println("sum = " + sum);

21 } catch (InterruptedException e) {

22 e.printStackTrace();

23 } catch (ExecutionException e) {

24 e.printStackTrace();

25 }

26

27 }

28 }

29

30

31 class MyCallable implements Callable<Integer> {

32 private int i = 0;

33

34 // 与run()方法不同的是，call()方法具有返回值

35 @Override

36 public Integer call() {

37 int sum = 0;

38 for (; i < 100; i++) {

39 System.out.println(Thread.currentThread().getName() + " " + i);

40 sum += i;

41 }

42 return sum;

43 }

44

45 }

[复制代码](javascript:void(0);)

首先，我们发现，在实现Callable接口中，此时不再是run()方法了，而是call()方法，此call()方法作为线程执行体，同时还具有返回值！在创建新的线程时，是通过FutureTask来包装MyCallable对象，同时作为了Thread对象的target。那么看下FutureTask类的定义：

1 public class FutureTask<V> implements RunnableFuture<V> {

2

3 //....

4

5 }

1 public interface RunnableFuture<V> extends Runnable, Future<V> {

2

3 void run();

4

5 }

于是，我们发现FutureTask类实际上是同时实现了Runnable和Future接口，由此才使得其具有Future和Runnable双重特性。通过Runnable特性，可以作为Thread对象的target，而Future特性，使得其可以取得新创建线程中的call()方法的返回值。

执行下此程序，我们发现sum = 4950永远都是最后输出的。而“主线程for循环执行完毕..”则很可能是在子线程循环中间输出。由CPU的线程调度机制，我们知道，“主线程for循环执行完毕..”的输出时机是没有任何问题的，那么为什么sum =4950会永远最后输出呢？

原因在于通过ft.get()方法获取子线程call()方法的返回值时，当子线程此方法还未执行完毕，ft.get()方法会一直阻塞，直到call()方法执行完毕才能取到返回值。

上述主要讲解了三种常见的线程创建方式，对于线程的启动而言，都是调用线程对象的start()方法，需要特别注意的是：**不能对同一线程对象两次调用start()方法。**

**三. Java多线程的就绪、运行和死亡状态**

就绪状态转换为运行状态：当此线程得到处理器资源；

运行状态转换为就绪状态：当此线程主动调用yield()方法或在运行过程中失去处理器资源。

运行状态转换为死亡状态：当此线程线程执行体执行完毕或发生了异常。

此处需要特别注意的是：当调用线程的yield()方法时，线程从运行状态转换为就绪状态，但接下来CPU调度就绪状态中的哪个线程具有一定的随机性，因此，可能会出现A线程调用了yield()方法后，接下来CPU仍然调度了A线程的情况。

由于实际的业务需要，常常会遇到需要在特定时机终止某一线程的运行，使其进入到死亡状态。目前最通用的做法是设置一boolean型的变量，当条件满足时，使线程执行体快速执行完毕。如：

[复制代码](javascript:void(0);)

1 public class ThreadTest {

2

3 public static void main(String[] args) {

4

5 MyRunnable myRunnable = new MyRunnable();

6 Thread thread = new Thread(myRunnable);

7

8 for (int i = 0; i < 100; i++) {

9 System.out.println(Thread.currentThread().getName() + " " + i);

10 if (i == 30) {

11 thread.start();

12 }

13 if(i == 40){

14 myRunnable.stopThread();

15 }

16 }

17 }

18 }

19

20 class MyRunnable implements Runnable {

21

22 private boolean stop;

23

24 @Override

25 public void run() {

26 for (int i = 0; i < 100 && !stop; i++) {

27 System.out.println(Thread.currentThread().getName() + " " + i);

28 }

29 }

30

31 public void stopThread() {

32 this.stop = true;

33 }

34

35 }

[复制代码](javascript:void(0);)