多线程的一些案例:

<https://www.cnblogs.com/neon/p/10918447.html>

java多线程传参的方式：

<https://www.cnblogs.com/jpfss/p/10783847.html>

在传统的同步开发模式下，当我们调用一个函数时，通过这个函数的参数将数据传入，并通过这个函数的返回值来返回最终的计算结果。但在多线程的异步开发模式下，数据的传递和返回和同步开发模式有很大的区别。由于线程的运行和结束是不可预料的，因此，在传递和返回数据时就无法象函数一样通过函数参数和return语句来返回数据。本文就以上原因介绍了几种用于向线程传递数据的方法，在下一篇文章中将介绍从线程中返回数据的方法。

    欲先取之，必先予之。一般在使用线程时都需要有一些初始化数据，然后线程利用这些数据进行加工处理，并返回结果。在这个过程中最先要做的就是向线程中传递数据。

**一、通过构造方法传递数据**

在创建线程时，必须要建立一个Thread类的或其子类的实例。因此，我们不难想到在调用start方法之前通过线程类的构造方法将数据传入线程。并将传入的数据使用类变量保存起来，以便线程使用(其实就是在run方法中使用)。下面的代码演示了如何通过构造方法来传递数据：

1. package mythread;
3. public class MyThread1 extends Thread
4. {
5. private String name;
7. public MyThread1(String name)
8. {
9. this.name = name;
10. }
11. public void run()
12. {
13. System.out.println("hello " + name);
14. }
15. public static void main(String[] args)
16. {
17. Thread thread = new MyThread1("world");
18. thread.start();
19. }
20. }

 由于这种方法是在创建线程对象的同时传递数据的，因此，在线程运行之前这些数据就就已经到位了，这样就不会造成数据在线程运行后才传入的现象。如果要传递更复杂的数据，可以使用集合、类等数据结构。使用构造方法来传递数据虽然比较安全，但如果要传递的数据比较多时，就会造成很多不便。由于Java没有默认参数，要想实现类似默认参数的效果，就得使用重载，这样不但使构造方法本身过于复杂，又会使构造方法在数量上大增。因此，要想避免这种情况，就得通过类方法或类变量来传递数据。

**二、通过变量和方法传递数据**

    向对象中传入数据一般有两次机会，第一次机会是在建立对象时通过构造方法将数据传入，另外一次机会就是在类中定义一系列的public的方法或变量（也可称之为字段）。然后在建立完对象后，通过对象实例逐个赋值。下面的代码是对MyThread1类的改版，使用了一个setName方法来设置name变量：

1. package mythread;
3. public class MyThread2 implements Runnable
4. {
5. private String name;
7. public void setName(String name)
8. {
9. this.name = name;
10. }
11. public void run()
12. {
13. System.out.println("hello " + name);
14. }
15. public static void main(String[] args)
16. {
17. MyThread2 myThread = new MyThread2();
18. myThread.setName("world");
19. Thread thread = new Thread(myThread);
20. thread.start();
21. }
22. }

**三、通过回调函数传递数据**

   上面讨论的两种向线程中传递数据的方法是最常用的。但这两种方法都是main方法中主动将数据传入线程类的。这对于线程来说，是被动接收这些数据的。然而，在有些应用中需要在线程运行的过程中动态地获取数据，如在下面代码的run方法中产生了3个随机数，然后通过Work类的process方法求这三个随机数的和，并通过Data类的value将结果返回。从这个例子可以看出，在返回value之前，必须要得到三个随机数。也就是说，这个value是无法事先就传入线程类的。

1. package mythread;
3. class Data
4. {
5. public int value = 0;
6. }
7. class Work
8. {
9. public void process(Data data, Integer numbers)
10. {
11. for (int n : numbers)
12. {
13. data.value += n;
14. }
15. }
16. }
17. public class MyThread3 extends Thread
18. {
19. private Work work;
21. public MyThread3(Work work)
22. {
23. this.work = work;
24. }
25. public void run()
26. {
27. java.util.Random random = new java.util.Random();
28. Data data = new Data();
29. int n1 = random.nextInt(1000);
30. int n2 = random.nextInt(2000);
31. int n3 = random.nextInt(3000);
32. work.process(data, n1, n2, n3); // 使用回调函数
33. System.out.println(String.valueOf(n1) + "+" + String.valueOf(n2) + "+"
34. + String.valueOf(n3) + "=" + data.value);
35. }
36. public static void main(String[] args)
37. {
38. Thread thread = new MyThread3(new Work());
39. thread.start();
40. }
41. }

在上面代码中的process方法被称为回调函数。从本质上说，回调函数就是事件函数。在Windows API中常使用回调函数和调用API的程序之间进行数据交互。因此，调用回调函数的过程就是最原始的引发事件的过程。在这个例子中调用了process方法来获得数据也就相当于在run方法中引发了一个事件。