[java多线程中的join方法详解](http://blog.csdn.net/fg2006/article/details/6393768)

方法Join是干啥用的？ 简单回答，同步，如何同步？ 怎么实现的？ 下面将逐个回答。

    自从接触**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "_blank)**多线程，一直对Join理解不了。JDK是这样说的：join public final void join（long millis）throws InterruptedException Waits at most millis milliseconds for this thread to die. A timeout of 0 means to wait forever.大家能理解吗？ 字面意思是等待一段时间直到这个线程死亡，我的疑问是那个线程，是它本身的线程还是调用它的线程的，上代码：

package concurrentstudy;  
/\*\*  
 \*  
 \* @author vma  
 \*/  
public class JoinTest {  
    public static void main(String[] args) {  
        Thread t = new Thread(new RunnableImpl());  
        t.start();  
        try {  
            t.join(1000);  
            System.out.println("joinFinish");  
        } catch (InterruptedException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
       
        }  
    }  
}  
class RunnableImpl implements Runnable {  
  
    @Override  
    public void run() {  
        try {  
            System.out.println("Begin sleep");  
            Thread.sleep(1000);  
           System.out.println("End sleep");  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
  
    }  
}

结果是：  
Begin sleep  
End sleep  
joinFinish  
明白了吧，当main线程调用t.join时，main线程等待t线程，等待时间是1000，如果t线程Sleep 2000呢   
 public void run() {  
        try {  
            System.out.println("Begin sleep");  
            // Thread.sleep(1000);  
            Thread.sleep(2000);  
           System.out.println("End sleep");  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
  
    }  
结果是：  
Begin sleep  
joinFinish  
End sleep  
也就是说main线程只等1000毫秒，不管T什么时候结束，如果是t.join()呢， 看代码：     
 public final void join() throws InterruptedException {  
    join(0);  
    }  
    就是说如果是t.join() = t.join(0) 0　JDK这样说的 A timeout of 0 means to wait forever 字面意思是永远等待，是这样吗？  
    其实是等到t结束后。  
    这个是怎么实现的吗？ 看JDK代码：

    /\*\*  
     \* Waits at most <code>millis</code> milliseconds for this thread to   
     \* die. A timeout of <code>0</code> means to wait forever.   
     \*  
     \* @param      millis   the time to wait in milliseconds.  
     \* @exception  InterruptedException if any thread has interrupted  
     \*             the current thread.  The <i>interrupted status</i> of the  
     \*             current thread is cleared when this exception is thrown.  
     \*/  
    public final synchronized void join(long millis)   
    throws InterruptedException {  
    long base = System.currentTimeMillis();  
    long now = 0;  
  
    if (millis < 0) {  
            throw new IllegalArgumentException("timeout value is negative");  
    }  
  
    if (millis == 0) {  
        while (isAlive()) {  
        wait(0);  
        }  
    } else {  
        while (isAlive()) {  
        long delay = millis - now;  
        if (delay <= 0) {  
            break;  
        }  
        wait(delay);  
        now = System.currentTimeMillis() - base;  
        }  
    }  
    }

    其实**Join方法实现是通过wait（小提示：Object 提供的方法）。 当main线程调用t.join时候，main线程会获得线程对象t的锁（wait 意味着拿到该对象的锁),调用该对象的wait(等待时间)，直到该对象唤醒main线程，比如退出后。**  
  
    这就意味着main 线程调用t.join时，必须能够拿到线程t对象的锁，如果拿不到它是无法wait的，刚开的例子t.join(1000)不是说明了main线程等待1 秒，如果在它等待之前，其他线程获取了t对象的锁，它等待时间可不就是1毫秒了。上代码介绍：

/\*  
 \* To change this template, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/  
package concurrentstudy;  
/\*\*  
 \*  
 \* @author vma  
 \*/  
public class JoinTest {  
    public static void main(String[] args) {  
        Thread t = new Thread(new RunnableImpl());  
       new ThreadTest(t).start();  
        t.start();  
        try {  
            t.join();  
            System.out.println("joinFinish");  
        } catch (InterruptedException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
       
        }  
    }  
}  
class ThreadTest extends Thread {  
  
    Thread thread;  
  
    public ThreadTest(Thread thread) {  
        this.thread = thread;  
    }  
  
    @Override  
    public void run() {  
        holdThreadLock();  
    }  
  
    public void holdThreadLock() {  
        synchronized (thread) {  
            System.out.println("getObjectLock");  
            try {  
                Thread.sleep(9000);  
  
            } catch (InterruptedException ex) {  
             ex.printStackTrace();  
            }  
            System.out.println("ReleaseObjectLock");  
        }  
  
    }  
}  
  
class RunnableImpl implements Runnable {  
  
    @Override  
    public void run() {  
        try {  
            System.out.println("Begin sleep");  
            Thread.sleep(2000);  
           System.out.println("End sleep");  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
  
  
    }  
}

    在main方法中 通过new ThreadTest(t).start();实例化ThreadTest 线程对象， 它在holdThreadLock()方法中，通过 synchronized (thread)，获取线程对象t的锁，并Sleep（9000）后释放，这就意味着，即使  
main方法t.join(1000),等待一秒钟，它必须等待ThreadTest 线程释放t锁后才能进入wait方法中，它实际等待时间是9000+1000 MS  
运行结果是：  
getObjectLock  
Begin sleep  
End sleep  
ReleaseObjectLock  
joinFinish

转自：http://java.chinaitlab.com/JDK/760879.html

ps：

**二、为什么要用join()方法**

主线程生成并起动了子线程，而子线程里要进行大量的耗时的运算(这里可以借鉴下线程的作用)，当主线程处理完其他的事务后，需要用到子线程的处理结果，这个时候就要用到join();方法了。

**三、join方法的作用**

在网上看到有人说“将两个线程合并”。这样解释我觉得理解起来还更麻烦。不如就借鉴下API里的说法：

“等待该线程终止。”

解释一下，是主线程(我在“一”里已经命名过了)等待子线程的终止。也就是在子线程调用了join()方法后面的代码，只有等到子线程结束了才能执行。(Waits for this thread to die.)

**四、用实例来理解**

写一个简单的例子来看一下join()的用法，一共三个类：

1.CustomThread 类

2. CustomThread1类

3. JoinTestDemo 类，main方法所在的类。

**代码1：**

1. **package** wxhx.csdn2;
2. /\*\*
3. \*
4. \* @author bzwm
5. \*
6. \*/
7. **class** CustomThread1 **extends** Thread {
8. **public** CustomThread1() {
9. **super**("[CustomThread1] Thread");
10. };
11. **public** **void** run() {
12. String threadName = Thread.currentThread().getName();
13. System.out.println(threadName + " start.");
14. **try** {
15. **for** (**int** i = 0; i < 5; i++) {
16. System.out.println(threadName + " loop at " + i);
17. Thread.sleep(1000);
18. }
19. System.out.println(threadName + " end.");
20. } **catch** (Exception e) {
21. System.out.println("Exception from " + threadName + ".run");
22. }
23. }
24. }
25. **class** CustomThread **extends** Thread {
26. CustomThread1 t1;
27. **public** CustomThread(CustomThread1 t1) {
28. **super**("[CustomThread] Thread");
29. **this**.t1 = t1;
30. }
31. **public** **void** run() {
32. String threadName = Thread.currentThread().getName();
33. System.out.println(threadName + " start.");
34. **try** {
35. t1.join();
36. System.out.println(threadName + " end.");
37. } **catch** (Exception e) {
38. System.out.println("Exception from " + threadName + ".run");
39. }
40. }
41. }
42. **public** **class** JoinTestDemo {
43. **public** **static** **void** main(String[] args) {
44. String threadName = Thread.currentThread().getName();
45. System.out.println(threadName + " start.");
46. CustomThread1 t1 = **new** CustomThread1();
47. CustomThread t = **new** CustomThread(t1);
48. **try** {
49. t1.start();
50. Thread.sleep(2000);
51. t.start();
52. t.join();//在代碼2里，將此處注釋掉
53. } **catch** (Exception e) {
54. System.out.println("Exception from main");
55. }
56. System.out.println(threadName + " end!");
57. }
58. }

**打印结果：**

main start.//main方法所在的线程起动，但没有马上结束，因为调用t.join();，所以要等到t结束了，此线程才能向下执行。

[CustomThread1] Thread start.//线程CustomThread1起动

[CustomThread1] Thread loop at 0//线程CustomThread1执行

[CustomThread1] Thread loop at 1//线程CustomThread1执行

[CustomThread] Thread start.//线程CustomThread起动，但没有马上结束，因为调用t1.join();，所以要等到t1结束了，此线程才能向下执行。

[CustomThread1] Thread loop at 2//线程CustomThread1继续执行

[CustomThread1] Thread loop at 3//线程CustomThread1继续执行

[CustomThread1] Thread loop at 4//线程CustomThread1继续执行

[CustomThread1] Thread end. //线程CustomThread1结束了

[CustomThread] Thread end.// 线程CustomThread在t1.join();阻塞处起动，向下继续执行的结果

main end!//线程CustomThread结束，此线程在t.join();阻塞处起动，向下继续执行的结果。

修改一下代码，得到**代码2：(这里只写出修改的部分)**

1. **public class JoinTestDemo {**
2. **public static void main(String[] args) {**
3. **String threadName = Thread.currentThread().getName();**
4. **System.out.println(threadName + " start.");**
5. **CustomThread1 t1 = new CustomThread1();**
6. **CustomThread t = new CustomThread(t1);**
7. **try {**
8. **t1.start();**
9. **Thread.sleep(2000);**
10. **t.start();**
11. **//          t.join();//在代碼2里，將此處注釋掉**
12. **} catch (Exception e) {**
13. **System.out.println("Exception from main");**
14. **}**
15. **System.out.println(threadName + " end!");**
16. **}**
17. **}**

**打印结果：**

main start. // main方法所在的线程起动，但没有马上结束，这里并不是因为join方法，而是因为Thread.sleep(2000);

[CustomThread1] Thread start. //线程CustomThread1起动

[CustomThread1] Thread loop at 0//线程CustomThread1执行

[CustomThread1] Thread loop at 1//线程CustomThread1执行

main end!// Thread.sleep(2000);结束，虽然在线程CustomThread执行了t1.join();，但这并不会影响到其他线程(这里main方法所在的线程)。

[CustomThread] Thread start. //线程CustomThread起动，但没有马上结束，因为调用t1.join();，所以要等到t1结束了，此线程才能向下执行。

[CustomThread1] Thread loop at 2//线程CustomThread1继续执行

[CustomThread1] Thread loop at 3//线程CustomThread1继续执行

[CustomThread1] Thread loop at 4//线程CustomThread1继续执行

[CustomThread1] Thread end. //线程CustomThread1结束了

[CustomThread] Thread end. // 线程CustomThread在t1.join();阻塞处起动，向下继续执行的结果

**五、从源码看join()方法**

在CustomThread的run方法里，执行了t1.join();，进入看一下它的JDK源码：

1. **public** **final** **void** join() **throws** InterruptedException {
2. n(0);
3. }

然后进入join(0)方法：

1. /\*\*
2. \* Waits at most <code>millis</code> milliseconds for this thread to
3. \* die. A timeout of <code>0</code> means to wait forever. //注意这句
4. \*
5. \* @param      millis   the time to wait in milliseconds.
6. \* @exception  InterruptedException if another thread has interrupted
7. \*             the current thread.  The <i>interrupted status</i> of the
8. \*             current thread is cleared when this exception is thrown.
9. \*/
10. **public** **final** **synchronized** **void** join(**long** millis) //参数millis为0.
11. **throws** InterruptedException {
12. **long** base = System.currentTimeMillis();
13. **long** now = 0;
14. **if** (millis < 0) {
15. **throw** **new** IllegalArgumentException("timeout value is negative");
16. }
17. **if** (millis == 0) {//进入这个分支
18. **while** (isAlive()) {//判断本线程是否为活动的。这里的本线程就是t1.
19. wait(0);//阻塞
20. }
21. } **else** {
22. **while** (isAlive()) {
23. **long** delay = millis - now;
24. **if** (delay <= 0) {
25. **break**;
26. }
27. wait(delay);
28. now = System.currentTimeMillis() - base;
29. }
30. }
31. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

单纯从代码上看，如果线程被生成了，但还未被起动，调用它的join()方法是没有作用的。将直接继续向下执行，这里就不写代码验证了。

转自：

http://blog.csdn[**.NET**](http://lib.csdn.net/base/dotnet)/bzwm/archive/2009/02/12/3881392.aspx