[**Java线程个数简单控制**](http://jiangzhengjun.iteye.com/blog/531472)

**博客分类：**

* [Java](http://jiangzhengjun.iteye.com/category/50932)

[Java](http://www.iteye.com/blogs/tag/Java)[thread](http://www.iteye.com/blogs/tag/thread)

系统在运行时，有时候我们要控制它同时允许多少个线程运行，如果太多可能引起内存溢出之类的异常，所以在线程比较多的情况下我们可以控制它的最大线程数，这样系统可以在一种比较稳定的状态下运行。下面是一个简单实现，可以少加修改用在系统中去。

**Java代码**



1. **package** thread;

4. **public** **class** CreateThread {
6. **public** **static** **void** main(String[] args) **throws** InterruptedException {
7. //比如我们现要创建30个子线程，但允许最大线程数为10
8. **for** (**int** i = 0; i < 30; i++) {
9. // 获取锁资源
10. **synchronized** (SubThread.**class**) {
11. **int** tCount = SubThread.getThreadCounts();
13. // 如果入库线程达到了最大允许的线程数
14. **while** (tCount >= 10) {
15. System.out.println("系统当前线程数为：" + tCount
16. + "，已达到最大线程数 10，请等待其他线程执行完毕并释放系统资源");
17. // 释放锁，等待“线程数”资源，等待其他入库线程执行完毕
18. SubThread.**class**.wait();
20. /\*
21. \* 带条件的wait()（即放在条件语句执行的wait()方法），一定要放在while
22. \* 条件里执行，因为被唤醒之后该线程有可能不会马上执行，只是把当前线程从等
23. \* 待池中放入对象锁池队列中，直到获取到锁后才开始运行。所以在同步块里wait方
24. \* 法前后的代码块不是原子性的，会分开执行，但sleep()不同，睡时不会释放锁。
25. \* 所以等到执行时，可能条件已被其他的线程给改变了（像这里的最大线程数），此
26. \* 时就需要再此判断该条件，否则条件可能不成立了
27. \*/
28. tCount = SubThread.getThreadCounts();
29. }
30. // 重新启动一个子线程
31. Thread td = **new** SubThread();
32. td.start();
33. System.out.println("已创建新的子线程: " + td.getName());
34. }
35. }
37. }
38. }

**Java代码**



1. **package** thread;
3. **public** **class** SubThread **extends** Thread {
5. // 线程计数器
6. **static** **private** **int** threadCounts;
8. // 线程名称池
9. **static** **private** String threadNames[];
10. **static** {
11. // 假设这里允许系统同时运行最大线程数为10个
12. **int** maxThreadCounts = 10;
13. threadNames = **new** String[maxThreadCounts];
14. // 初始化线程名称池
15. **for** (**int** i = 1; i <= maxThreadCounts; i++) {
16. threadNames[i - 1] = "子线程\_" + i;
17. }
18. }
20. **public** SubThread() {
21. // 临界资源锁定
22. **synchronized** (SubThread.**class**) {
23. // 线程总数加1
24. threadCounts++;
26. // 从线程名称池中取出一个未使用的线程名
27. **for** (**int** i = 0; i < threadNames.length; i++) {
28. **if** (threadNames[i] != **null**) {
29. String temp = threadNames[i];
30. // 名被占用后清空
31. threadNames[i] = **null**;
32. // 初始化线程名称
33. **this**.setName(temp);
34. **break**;
35. }
36. }
37. }
38. }
40. **public** **void** run() {
41. System.out.println("-->>" + **this**.getName() + "开始运行");
42. **try** {
43. //模拟程序耗时
44. Thread.sleep(1000);
45. // ...
46. } **catch** (Exception e) {
47. System.out.println(e);
48. } **finally** {
49. **synchronized** (SubThread.**class**) {
51. // 线程运行完毕后减1
52. threadCounts--;
54. // 释放线程名称
55. String[] threadName = **this**.getName().split("\_");
57. // 线程名使用完后放入名称池
58. threadNames[Integer.parseInt(threadName[1]) - 1] = **this**.getName();
60. /\*
61. \* 通知其他被阻塞的线程，但如果其他线程要执行，则该同步块一定要运行结束（即直
62. \* 到释放占的锁），其他线程才有机会执行，所以这里的只是唤醒在此对象监视器上等待
63. \* 的所有线程，让他们从等待池中进入对象锁池队列中，而这些线程重新运行时它们一定
64. \* 要先要得该锁后才可能执行，这里的notifyAll是不会释放锁的，试着把下面的睡眠语
65. \* 句注释去掉，即使你已调用了notify方法，发现CreateThread中的同步块还是好
66. \* 像一直处于对象等待状态，其实调用notify方法后，CreateThread线程进入了对象锁
67. \* 池队列中了，只要它一获取到锁，CreateThread所在线程就会真真的被唤醒并运行。
68. \*/
69. SubThread.**class**.notifyAll();
70. //              try {
71. //                  Thread.sleep(10000);
72. //              } catch (InterruptedException e) {
73. //                  e.printStackTrace();
74. //              }
75. System.out.println("----" + **this**.getName() + " 所占用资源释放完毕，当前系统正在运行的子线程数："
76. + threadCounts);
77. }
78. System.out.println("<<--" + **this**.getName() + "运行结束");
79. }
80. }
82. **static** **public** **int** getThreadCounts() {
83. **synchronized** (SubThread.**class**) {
84. **return** threadCounts;
85. }
86. }
87. }

**某次运行结果：**

已创建新的子线程: 子线程\_1  
已创建新的子线程: 子线程\_2  
已创建新的子线程: 子线程\_3  
-->>子线程\_2开始运行  
已创建新的子线程: 子线程\_4  
-->>子线程\_4开始运行  
已创建新的子线程: 子线程\_5  
已创建新的子线程: 子线程\_6  
-->>子线程\_6开始运行  
已创建新的子线程: 子线程\_7  
已创建新的子线程: 子线程\_8  
已创建新的子线程: 子线程\_9  
已创建新的子线程: 子线程\_10  
系统当前线程数为：10，已达到最大线程数 10，请等待其他线程执行完毕并释放系统资源  
-->>子线程\_1开始运行  
-->>子线程\_8开始运行  
-->>子线程\_10开始运行  
-->>子线程\_3开始运行  
-->>子线程\_5开始运行  
-->>子线程\_7开始运行  
-->>子线程\_9开始运行  
----子线程\_2 所占用资源释放完毕，当前系统正在运行的子线程数：9  
已创建新的子线程: 子线程\_2  
系统当前线程数为：10，已达到最大线程数 10，请等待其他线程执行完毕并释放系统资源  
<<--子线程\_2运行结束  
-->>子线程\_2开始运行  
----子线程\_4 所占用资源释放完毕，当前系统正在运行的子线程数：9  
<<--子线程\_4运行结束  
已创建新的子线程: 子线程\_4  
系统当前线程数为：10，已达到最大线程数 10，请等待其他线程执行完毕并释放系统资源  
-->>子线程\_4开始运行  
----子线程\_6 所占用资源释放完毕，当前系统正在运行的子线程数：9  
<<--子线程\_6运行结束  
----子线程\_10 所占用资源释放完毕，当前系统正在运行的子线程数：8  
<<--子线程\_10运行结束  
----子线程\_8 所占用资源释放完毕，当前系统正在运行的子线程数：7  
<<--子线程\_8运行结束  
已创建新的子线程: 子线程\_6  
已创建新的子线程: 子线程\_8  
已创建新的子线程: 子线程\_10  
系统当前线程数为：10，已达到最大线程数 10，请等待其他线程执行完毕并释放系统资源  
----子线程\_1 所占用资源释放完毕，当前系统正在运行的子线程数：9  
<<--子线程\_1运行结束  
-->>子线程\_6开始运行  
-->>子线程\_10开始运行  
已创建新的子线程: 子线程\_1  
----子线程\_7 所占用资源释放完毕，当前系统正在运行的子线程数：9  
<<--子线程\_7运行结束  
----子线程\_9 所占用资源释放完毕，当前系统正在运行的子线程数：8  
<<--子线程\_9运行结束  
-->>子线程\_8开始运行  
-->>子线程\_1开始运行  
----子线程\_5 所占用资源释放完毕，当前系统正在运行的子线程数：7  
<<--子线程\_5运行结束  
----子线程\_3 所占用资源释放完毕，当前系统正在运行的子线程数：6  
<<--子线程\_3运行结束  
已创建新的子线程: 子线程\_3  
已创建新的子线程: 子线程\_5  
-->>子线程\_5开始运行  
已创建新的子线程: 子线程\_7  
已创建新的子线程: 子线程\_9  
系统当前线程数为：10，已达到最大线程数 10，请等待其他线程执行完毕并释放系统资源  
-->>子线程\_9开始运行  
-->>子线程\_3开始运行  
-->>子线程\_7开始运行  
----子线程\_4 所占用资源释放完毕，当前系统正在运行的子线程数：9  
<<--子线程\_4运行结束  
----子线程\_8 所占用资源释放完毕，当前系统正在运行的子线程数：8  
<<--子线程\_8运行结束  
----子线程\_1 所占用资源释放完毕，当前系统正在运行的子线程数：7  
<<--子线程\_1运行结束  
已创建新的子线程: 子线程\_1  
已创建新的子线程: 子线程\_4  
-->>子线程\_4开始运行  
已创建新的子线程: 子线程\_8  
-->>子线程\_1开始运行  
-->>子线程\_8开始运行  
----子线程\_5 所占用资源释放完毕，当前系统正在运行的子线程数：9  
<<--子线程\_5运行结束  
已创建新的子线程: 子线程\_5  
----子线程\_2 所占用资源释放完毕，当前系统正在运行的子线程数：9  
<<--子线程\_2运行结束  
----子线程\_6 所占用资源释放完毕，当前系统正在运行的子线程数：8  
<<--子线程\_6运行结束  
-->>子线程\_5开始运行  
----子线程\_10 所占用资源释放完毕，当前系统正在运行的子线程数：7  
<<--子线程\_10运行结束  
----子线程\_9 所占用资源释放完毕，当前系统正在运行的子线程数：6  
<<--子线程\_9运行结束  
已创建新的子线程: 子线程\_2  
已创建新的子线程: 子线程\_6  
已创建新的子线程: 子线程\_9  
已创建新的子线程: 子线程\_10  
系统当前线程数为：10，已达到最大线程数 10，请等待其他线程执行完毕并释放系统资源  
-->>子线程\_6开始运行  
-->>子线程\_10开始运行  
----子线程\_3 所占用资源释放完毕，当前系统正在运行的子线程数：9  
已创建新的子线程: 子线程\_3  
系统当前线程数为：10，已达到最大线程数 10，请等待其他线程执行完毕并释放系统资源  
<<--子线程\_3运行结束  
----子线程\_7 所占用资源释放完毕，当前系统正在运行的子线程数：9  
已创建新的子线程: 子线程\_7  
<<--子线程\_7运行结束  
-->>子线程\_2开始运行  
-->>子线程\_9开始运行  
-->>子线程\_7开始运行  
-->>子线程\_3开始运行  
----子线程\_4 所占用资源释放完毕，当前系统正在运行的子线程数：9  
<<--子线程\_4运行结束  
----子线程\_1 所占用资源释放完毕，当前系统正在运行的子线程数：8  
<<--子线程\_1运行结束  
----子线程\_8 所占用资源释放完毕，当前系统正在运行的子线程数：7  
<<--子线程\_8运行结束  
----子线程\_5 所占用资源释放完毕，当前系统正在运行的子线程数：6  
<<--子线程\_5运行结束  
----子线程\_6 所占用资源释放完毕，当前系统正在运行的子线程数：5  
<<--子线程\_6运行结束  
----子线程\_10 所占用资源释放完毕，当前系统正在运行的子线程数：4  
<<--子线程\_10运行结束  
----子线程\_2 所占用资源释放完毕，当前系统正在运行的子线程数：3  
<<--子线程\_2运行结束  
----子线程\_7 所占用资源释放完毕，当前系统正在运行的子线程数：2  
<<--子线程\_7运行结束  
----子线程\_9 所占用资源释放完毕，当前系统正在运行的子线程数：1  
<<--子线程\_9运行结束  
----子线程\_3 所占用资源释放完毕，当前系统正在运行的子线程数：0  
<<--子线程\_3运行结束