  在并发环境下，解决共享资源冲突问题时，可以考虑使用锁机制。

**1. 对象锁**

      所有对象都自动含有单一的锁。  
     JVM负责跟踪对象被加锁的次数。如果一个对象被解锁，其计数变为0。在任务（线程）第一次给对象加锁的时候，计数变为1。每当这个相同的任务（线程）在此对象上获得锁时，计数会递增。  
     只有首先获得锁的任务（线程）才能继续获取该对象上的多个锁。  
     每当任务离开一个synchronized方法，计数递减，当计数为0的时候，锁被完全释放，此时别的任务就可以使用此资源。

**2. 类锁**

      对于同步静态方法/静态变量互斥体，由于一个class不论被实例化多少次，其中的静态方法和静态变量在内存中都只由一份。所以，一旦一个静态的方法被申明为synchronized。此类所有的实例化对象在调用此方法，共用同一把锁，我们称之为类锁。一旦一个静态变量被作为synchronized block的互斥体。进入此同步区域时，都要先获得此静态变量的对象锁。

      由上述同步静态方法引申出一个概念，那就是类锁。其实系统中并不存在什么类锁。当一个同步静态方法被调用时，系统获取的其实就是代表该类的类对象的对象锁  
  
      可以尝试用以下方式获取类锁

**[java]** [view plain](http://blog.csdn.net/ymeng_bupt/article/details/6826936) [copy](http://blog.csdn.net/ymeng_bupt/article/details/6826936)

1. **synchronized** (xxx.**class**) {...}
3. **synchronized** (Class.forName("xxx")) {...}

       若要同时获取两种锁，同时获取类锁和对象锁是允许的，并不会产生任何问题，但使用类锁时一定要注意，一旦产生类锁的嵌套获取的话，就会产生死锁，因为每个class在内存中都只能生成一个Class实例对象。

**3. synchronized同步块**

**3.1. 同步到单一对象锁**

        当使用同步块时，如果方法下的同步块都同步到一个对象上的锁，则所有的任务（线程）只能互斥的进入这些同步块。  
        Resource1.[**Java**](http://lib.csdn.net/base/java)演示了三个线程（包括main线程）试图进入某个类的三个不同的方法的同步块中，虽然这些同步块处在不同的方法中，但由于是同步到同一个对象（当前对象 synchronized (this)），所以对它们的方法依然是互斥的。  
Resource1.java

**[java]** [view plain](http://blog.csdn.net/ymeng_bupt/article/details/6826936) [copy](http://blog.csdn.net/ymeng_bupt/article/details/6826936)

1. **package** com.zj.lock;
2. **import** java.util.concurrent.TimeUnit;
4. **public** **class** Resource1 {
5. **public** **void** f() {
6. // other operations should not be locked...
7. System.out.println(Thread.currentThread().getName()
8. + ":not synchronized in f()");
9. **synchronized** (**this**) {
10. **for** (**int** i = 0; i < 5; i++) {
11. System.out.println(Thread.currentThread().getName()
12. + ":synchronized in f()");
13. **try** {
14. TimeUnit.SECONDS.sleep(3);
15. } **catch** (InterruptedException e) {
16. e.printStackTrace();
17. }
18. }
19. }
20. }
22. **public** **void** g() {
23. // other operations should not be locked...
24. System.out.println(Thread.currentThread().getName()
25. + ":not synchronized in g()");
26. **synchronized** (**this**) {
27. **for** (**int** i = 0; i < 5; i++) {
28. System.out.println(Thread.currentThread().getName()
29. + ":synchronized in g()");
30. **try** {
31. TimeUnit.SECONDS.sleep(3);
32. } **catch** (InterruptedException e) {
33. e.printStackTrace();
34. }
35. }
36. }
37. }
39. **public** **void** h() {
40. // other operations should not be locked...
41. System.out.println(Thread.currentThread().getName()
42. + ":not synchronized in h()");
43. **synchronized** (**this**) {
44. **for** (**int** i = 0; i < 5; i++) {
45. System.out.println(Thread.currentThread().getName()
46. + ":synchronized in h()");
47. **try** {
48. TimeUnit.SECONDS.sleep(3);
49. } **catch** (InterruptedException e) {
50. e.printStackTrace();
51. }
52. }
53. }
54. }
56. **public** **static** **void** main(String[] args) {
57. **final** Resource1 rs = **new** Resource1();
59. **new** Thread() {
60. **public** **void** run() {
61. rs.f();
62. }
63. }.start();
65. **new** Thread() {
66. **public** **void** run() {
67. rs.g();
68. }
69. }.start();
71. rs.h();
72. }
73. }

结果：  
Thread-0:not synchronized in f()  
Thread-0:synchronized in f()  
main:not synchronized in h()  
Thread-1:not synchronized in g()  
Thread-0:synchronized in f()  
Thread-0:synchronized in f()  
Thread-0:synchronized in f()  
Thread-0:synchronized in f()  
Thread-1:synchronized in g()  
Thread-1:synchronized in g()  
Thread-1:synchronized in g()  
Thread-1:synchronized in g()  
Thread-1:synchronized in g()  
main:synchronized in h()  
main:synchronized in h()  
main:synchronized in h()  
main:synchronized in h()  
main:synchronized in h()

**3.2. 同步到多个对象锁**

        Resource1.java演示了三个线程（包括main线程）试图进入某个类的三个不同的方法的同步块中，这些同步块处在不同的方法中，并且是同步到三个不同的对象（synchronized (this)，synchronized(syncObject1)，synchronized (syncObject2)），所以对它们的方法中的临界资源访问是独立的。  
Resource2.java

**[java]** [view plain](http://blog.csdn.net/ymeng_bupt/article/details/6826936) [copy](http://blog.csdn.net/ymeng_bupt/article/details/6826936)

1. **package** com.zj.lock;
2. **import** java.util.concurrent.TimeUnit;
4. **public** **class** Resource2 {
5. **private** Object syncObject1 = **new** Object();
6. **private** Object syncObject2 = **new** Object();
8. **public** **void** f() {
9. // other operations should not be locked...
10. System.out.println(Thread.currentThread().getName()
11. + ":not synchronized in f()");
12. **synchronized** (**this**) {
13. **for** (**int** i = 0; i < 5; i++) {
14. System.out.println(Thread.currentThread().getName()
15. + ":synchronized in f()");
16. **try** {
17. TimeUnit.SECONDS.sleep(3);
18. } **catch** (InterruptedException e) {
19. e.printStackTrace();
20. }
21. }
22. }
23. }
25. **public** **void** g() {
26. // other operations should not be locked...
27. System.out.println(Thread.currentThread().getName()
28. + ":not synchronized in g()");
29. **synchronized** (syncObject1) {
30. **for** (**int** i = 0; i < 5; i++) {
31. System.out.println(Thread.currentThread().getName()
32. + ":synchronized in g()");
33. **try** {
34. TimeUnit.SECONDS.sleep(3);
35. } **catch** (InterruptedException e) {
36. e.printStackTrace();
37. }
38. }
39. }
40. }
42. **public** **void** h() {
43. // other operations should not be locked...
44. System.out.println(Thread.currentThread().getName()
45. + ":not synchronized in h()");
46. **synchronized** (syncObject2) {
47. **for** (**int** i = 0; i < 5; i++) {
48. System.out.println(Thread.currentThread().getName()
49. + ":synchronized in h()");
50. **try** {
51. TimeUnit.SECONDS.sleep(3);
52. } **catch** (InterruptedException e) {
53. e.printStackTrace();
54. }
55. }
56. }
57. }
59. **public** **static** **void** main(String[] args) {
60. **final** Resource2 rs = **new** Resource2();
62. **new** Thread() {
63. **public** **void** run() {
64. rs.f();
65. }
66. }.start();
68. **new** Thread() {
69. **public** **void** run() {
70. rs.g();
71. }
72. }.start();
74. rs.h();
75. }
76. }

结果：  
Thread-0:not synchronized in f()  
Thread-0:synchronized in f()  
main:not synchronized in h()  
main:synchronized in h()  
Thread-1:not synchronized in g()  
Thread-1:synchronized in g()  
Thread-0:synchronized in f()  
main:synchronized in h()  
Thread-1:synchronized in g()  
Thread-0:synchronized in f()  
main:synchronized in h()  
Thread-1:synchronized in g()  
Thread-0:synchronized in f()  
main:synchronized in h()  
Thread-1:synchronized in g()  
Thread-0:synchronized in f()  
main:synchronized in h()  
Thread-1:synchronized in g()

**4. Lock对象锁**

      除了使用synchronized外，还可以使用Lock对象来创建临界区。Resource3.java的演示效果同Resource1.java；Resource4.java的演示效果同Resource2.java。  
Resource3.java

**[java]** [view plain](http://blog.csdn.net/ymeng_bupt/article/details/6826936) [copy](http://blog.csdn.net/ymeng_bupt/article/details/6826936)

1. **package** com.zj.lock;
2. **import** java.util.concurrent.TimeUnit;
3. **import** java.util.concurrent.locks.Lock;
4. **import** java.util.concurrent.locks.ReentrantLock;
6. **public** **class** Resource3 {
7. **private** Lock lock = **new** ReentrantLock();
9. **public** **void** f() {
10. // other operations should not be locked...
11. System.out.println(Thread.currentThread().getName()
12. + ":not synchronized in f()");
13. lock.lock();
14. **try** {
15. **for** (**int** i = 0; i < 5; i++) {
16. System.out.println(Thread.currentThread().getName()
17. + ":synchronized in f()");
18. **try** {
19. TimeUnit.SECONDS.sleep(3);
20. } **catch** (InterruptedException e) {
21. e.printStackTrace();
22. }
23. }
24. } **finally** {
25. lock.unlock();
26. }
27. }
29. **public** **void** g() {
30. // other operations should not be locked...
31. System.out.println(Thread.currentThread().getName()
32. + ":not synchronized in g()");
33. lock.lock();
34. **try** {
35. **for** (**int** i = 0; i < 5; i++) {
36. System.out.println(Thread.currentThread().getName()
37. + ":synchronized in g()");
38. **try** {
39. TimeUnit.SECONDS.sleep(3);
40. } **catch** (InterruptedException e) {
41. e.printStackTrace();
42. }
43. }
44. } **finally** {
45. lock.unlock();
46. }
47. }
49. **public** **void** h() {
50. // other operations should not be locked...
51. System.out.println(Thread.currentThread().getName()
52. + ":not synchronized in h()");
53. lock.lock();
54. **try** {
55. **for** (**int** i = 0; i < 5; i++) {
56. System.out.println(Thread.currentThread().getName()
57. + ":synchronized in h()");
58. **try** {
59. TimeUnit.SECONDS.sleep(3);
60. } **catch** (InterruptedException e) {
61. e.printStackTrace();
62. }
63. }
64. } **finally** {
65. lock.unlock();
66. }
67. }
69. **public** **static** **void** main(String[] args) {
70. **final** Resource3 rs = **new** Resource3();
72. **new** Thread() {
73. **public** **void** run() {
74. rs.f();
75. }
76. }.start();
78. **new** Thread() {
79. **public** **void** run() {
80. rs.g();
81. }
82. }.start();
84. rs.h();
85. }

结果：  
Thread-0:not synchronized in f()  
Thread-0:synchronized in f()  
main:not synchronized in h()  
Thread-1:not synchronized in g()  
Thread-0:synchronized in f()  
Thread-0:synchronized in f()  
Thread-0:synchronized in f()  
Thread-0:synchronized in f()  
main:synchronized in h()  
main:synchronized in h()  
main:synchronized in h()  
main:synchronized in h()  
main:synchronized in h()  
Thread-1:synchronized in g()  
Thread-1:synchronized in g()  
Thread-1:synchronized in g()  
Thread-1:synchronized in g()  
Thread-1:synchronized in g()

Resource4.java

**[java]** [view plain](http://blog.csdn.net/ymeng_bupt/article/details/6826936) [copy](http://blog.csdn.net/ymeng_bupt/article/details/6826936)

1. **package** com.zj.lock;
2. **import** java.util.concurrent.TimeUnit;
3. **import** java.util.concurrent.locks.Lock;
4. **import** java.util.concurrent.locks.ReentrantLock;
6. **public** **class** Resource4 {
7. **private** Lock lock1 = **new** ReentrantLock();
8. **private** Lock lock2 = **new** ReentrantLock();
9. **private** Lock lock3 = **new** ReentrantLock();
11. **public** **void** f() {
12. // other operations should not be locked...
13. System.out.println(Thread.currentThread().getName()
14. + ":not synchronized in f()");
15. lock1.lock();
16. **try** {
17. **for** (**int** i = 0; i < 5; i++) {
18. System.out.println(Thread.currentThread().getName()
19. + ":synchronized in f()");
20. **try** {
21. TimeUnit.SECONDS.sleep(3);
22. } **catch** (InterruptedException e) {
23. e.printStackTrace();
24. }
25. }
26. } **finally** {
27. lock1.unlock();
28. }
29. }
31. **public** **void** g() {
32. // other operations should not be locked...
33. System.out.println(Thread.currentThread().getName()
34. + ":not synchronized in g()");
35. lock2.lock();
36. **try** {
37. **for** (**int** i = 0; i < 5; i++) {
38. System.out.println(Thread.currentThread().getName()
39. + ":synchronized in g()");
40. **try** {
41. TimeUnit.SECONDS.sleep(3);
42. } **catch** (InterruptedException e) {
43. e.printStackTrace();
44. }
45. }
46. } **finally** {
47. lock2.unlock();
48. }
49. }
51. **public** **void** h() {
52. // other operations should not be locked...
53. System.out.println(Thread.currentThread().getName()
54. + ":not synchronized in h()");
55. lock3.lock();
56. **try** {
57. **for** (**int** i = 0; i < 5; i++) {
58. System.out.println(Thread.currentThread().getName()
59. + ":synchronized in h()");
60. **try** {
61. TimeUnit.SECONDS.sleep(3);
62. } **catch** (InterruptedException e) {
63. e.printStackTrace();
64. }
65. }
66. } **finally** {
67. lock3.unlock();
68. }
69. }
71. **public** **static** **void** main(String[] args) {
72. **final** Resource4 rs = **new** Resource4();
74. **new** Thread() {
75. **public** **void** run() {
76. rs.f();
77. }
78. }.start();
80. **new** Thread() {
81. **public** **void** run() {
82. rs.g();
83. }
84. }.start();
86. rs.h();
87. }
88. }

结果：  
Thread-0:not synchronized in f()  
Thread-0:synchronized in f()  
main:not synchronized in h()  
main:synchronized in h()  
Thread-1:not synchronized in g()  
Thread-1:synchronized in g()  
Thread-0:synchronized in f()  
main:synchronized in h()  
Thread-1:synchronized in g()  
Thread-0:synchronized in f()  
main:synchronized in h()  
Thread-1:synchronized in g()  
Thread-0:synchronized in f()  
main:synchronized in h()  
Thread-1:synchronized in g()  
Thread-0:synchronized in f()  
main:synchronized in h()

Thread-1:synchronized in g()

     另外，ReentrantLock可定时和可轮询的锁获取模式由tryLock方法实现。

**[java]** [view plain](http://blog.csdn.net/ymeng_bupt/article/details/6826936) [copy](http://blog.csdn.net/ymeng_bupt/article/details/6826936)

1. **public** **boolean** tryLock(); //等同于tryLock(0, TimeUnit.SECONDS),不停询问是否可获取锁
2. **public** **boolean** tryLock(**long** timeout,
3. TimeUnit unit)
4. **throws** InterruptedException    //timeout - 等待锁的时间,unit - timeout 参数的时间单位

**5. synchronized和lock的区别:**

      Lock 的锁定是通过代码实现的，而 synchronized 是在 JVM 层面上实现的。

      synchronized 在锁定时如果方法块抛出异常，JVM 会自动将锁释放掉，不会因为出了异常没有释放锁造成线程死锁。但是 Lock 的话就享受不到 JVM 带来自动的功能，出现异常时必须在 finally 将锁释放掉，否则将会引起死锁。

      在资源竞争不是很激烈的情况下，偶尔会有同步的情形下，synchronized是很合适的。原因在于，编译程序通常会尽可能的进行优化synchronize，另外可读性非常好，不管用没用过5.0多线程包的程序员都能理解。

      ReentrantLock:

      ReentrantLock提供了多样化的同步，比如有时间限制的同步，可以被Interrupt的同步（synchronized的同步是不能Interrupt的）等。在资源竞争不激烈的情形下，性能稍微比synchronized差点点。但是当同步非常激烈的时候，synchronized的性能一下子能下降好几十倍。而ReentrantLock确还能维持常态。

      Atomic:

      和上面的类似，不激烈情况下，性能比synchronized略逊，而激烈的时候，也能维持常态。激烈的时候，Atomic的性能会优于ReentrantLock一倍左右。但是其有一个缺点，就是只能同步一个值，一段代码中只能出现一个Atomic的变量，多于一个同步无效。因为他不能在多个Atomic之间同步。

      关于synchronized和lock的详细区别请看<http://www.ibm.com/developerworks/cn/java/j-jtp10264/index.html>