摆渡人，外婆桥！

----只要路选对了，就不怕有多远！

synchronized的作用

synchronized的作用

一、同步方法

public synchronized void methodAAA(){

//....

}

锁定的是调用这个同步方法的对象

测试：

a、不使用这个关键字修饰方法，两个线程调用同一个对象的这个方法。

目标类：

```
1⊟public class TestThread {
2⊟    public  void execute(){  //synchronized,未修饰
3⊟        for(int i=0;i<100;i++){
4 |           System.out.println(i);
5 ├       }
6 ├   }
7 └}
```

线程类：

```
1⊟public class ThreadA implements Runnable{
2 |    TestThread test=null;
3⊟    public ThreadA(TestThread pTest){  //对象有外部引入，这样保证是同一个对象
4 |        test=pTest;
5 ├   }
6⊟    public void run() {
7 |        test.execute();
```

< 2007年5月 >

日	一	二	三	四	五	六
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

公告

导航

- BlogJava
- 首页
- 发新随笔
- 发新文章
- 联系
- 聚合XML
- 管理

统计

- 随笔： 51
- 文章： 1
- 评论： 43
- 引用： 0

常用链接

- 我的随笔

```
8 ├    }
9 └}
```

调用：

```
1  TestThread test=new TestThread();
2  Runnable runabble=new ThreadA(test);
3  Thread a=new Thread(runabble,"A");
4  a.start();
5  Thread b=new Thread(runabble,"B");
6  b.start();
```

结果：
输出的数字交错在一起。说明不是同步的，两个方法在不同的线程中是异步调用的。

b、修改目标类，增加synchronized修饰

```
1⊟public class TestThread {
2⊟    public synchronized  void execute(){  //synchronized修饰
3⊟        for(int i=0;i<100;i++){
4 |           System.out.println(i);
5 ├       }
6 ├   }
7 └}
```

结果：
输出的数字是有序的，首先输出A的数字，然后是B，说明是同步的，虽然是不同的线程，但两个方法是同步调用的。
注意：上面虽然是两个不同的线程，但是是同一个实例对象。下面使用不同的实例对象进行测试。

c、每个线程都有独立的TestThread对象。
目标类：

```
1⊟public class TestThread {
2⊟    public synchronized void execute(){  //synchronized修饰
3⊟        for(int i=0;i<100;i++){
4 |           System.out.println(i);
5 ├       }
6 ├   }
7 └}
```


- 我的评论
- 我的参与
- 最新评论

留言簿(4)

- 给我留言
- 查看公开留言
- 查看私人留言

随笔分类

- JAVA(21) (rss)
- javascript&DHTML&CSS(15) (rss)
- Oracle/MySQL(6) (rss)
- SOA(1) (rss)
- WEB2.0(4) (rss)
- WorkFlow&BPEL(1) (rss)
- 名词概念(1) (rss)
- 架构模式 (rss)
- 汽车&甲壳虫 (rss)
- 设计模式(3) (rss)
- 软件工程(1) (rss)

随笔档案

- 2007年9月 (1)
- 2007年8月 (3)
- 2007年7月 (1)
- 2007年5月 (2)
- 2007年4月 (4)
- 2007年3月 (1)
- 2007年2月 (1)
- 2006年12月 (1)
- 2006年11月 (2)
- 2006年10月 (1)
- 2006年8月 (1)
- 2006年4月 (1)
- 2006年3月 (2)
- 2006年2月 (4)
- 2005年12月 (26)

线程类：

```
public class ThreadA implements Runnable{
    public void run() {
        TestThread test=new TestThread();
        test.execute();
    }
}

```

调用：

```
Runnable runabble=new ThreadA();
Thread a=new Thread(runabble,"A");
a.start();
Thread b=new Thread(runabble,"B");
b.start();
```

结果：
输出的数字交错在一起。说明虽然增加了synchronized 关键字来修饰方法，但是不同的线程调用各自的对象实例，两个方法仍然是异步的。

引申：
对于这种多个实例，要想实现同步即输出的数字是有序并且按线程先后顺序输出，我们可以增加一个静态变量，对它进行加锁(后面将说明锁定的对象)。

修改目标类：

```
public class TestThread {
    private static Object lock=new Object(); //必须是静态的。
    public  void execute(){
        synchronized(lock){
            for(int i=0;i<100;i++){
                System.out.println(i);
            }
        }
    }
}
```

二、同步代码块

```
public void method(SomeObject so){
    synchronized(so)
```

```
3 |       //.....
4 └    }
5  }
```

锁定一个对象，其实锁定的是该对象的引用（object reference）
谁拿到这个锁谁就可以运行它所控制的那段代码。当有一个明确的对象作为锁时，就可以按上面的代码写程序，但当没有明确的对象作为锁，只是想让一段代码同步时，可以创建一个特殊的instance变量（它必须是一个对象）来充当锁（上面的解决方法就是增加了一个状态锁）。

a、锁定一个对象，它不是静态的
private byte[] lock = new byte[0]; // 特殊的instance变量
目标类：

```
1⊟public class TestThread {
2 |    private Object lock=new Object();
3⊟    public  void execute(){
4⊟       synchronized(lock){  //增加了个锁，锁定了对象lock，在同一个类实例中，是线程安全的，但不同的实例还是不安全的。
5 |
6 |因为不同的实例有不同对象锁lock
7⊟          for(int i=0;i<100;i++){
8 |             System.out.println(i);
9 ├          }
10├       }
11├    }
12└}
```

其实上面锁定一个方法，等同于下面的：

```
1⊟public void execute(){
2⊟    synchronized(this){   //同步的是当然对象
3⊟       for(int i=0;i<100;i++){
4 |          System.out.println(i);
5 ├       }
6 ├    }
7 └}
```

b、锁定一个对象或方法，它是静态的
这样锁定，它锁定的是对象所属的类
public synchronized static void execute(){
  //...

--sheng

- 3. re: 可恶的“本页不但包含安全的内容，也包含不安全的内容。是否显示不安全的内容”对话框？
- 我的系统是直接使用<FRAME SRC='XXX.HTML'>
  这样的，照样出现提示。不知道为何？
- --Quentin
- 4. re: 可恶的“本页不但包含安全的内容，也包含不安全的内容。是否显示不安全的内容”对话框？
- 在firefox下，试试src="javascript:'about:blank' " 呢？
- --lin
- 5. re: synchronized的作用[未登录]
- 评论内容较长,点击标题查看
- --swingboat

阅读排行榜

- 1. synchronized的作用(10670)
- 2. 可恶的“本页不但包含安全的内容，也包含不安全的内容。是否显示不安全的内容”对话框？(4816)
- 3. javascript转换日期字符串为Date对象(4201)
- 4. 不同时区之间，时间的转换？(2174)
- 5. Table中table-layout:fixed样式的作用。(1614)

评论排行榜

- 1. synchronized的作用(15)
- 2. javascript在IE和Firefox中的区别1(6)
- 3. 可恶的“本页不但包含安全的内容，也包含不安全的内容。是否显示不安全的内容”对话框？(4)
- 4. 在IE下，页面只有一个text的时候，回车将自动submit。(3)
- 5. 不同时区之间，时间的转换？(3)

}

等同于

```
public class TestThread {
    public static void execute(){
        synchronized(TestThread.class){
            //…
        }
    }
}
```

测试：

目标类：

```
public class TestThread {
    private static Object lock=new Object();
    public synchronized static void execute(){  //同步静态方法
        for(int i=0;i<100;i++){
            System.out.println(i);
        }
    }
    public static void execute1(){
        for(int i=0;i<100;i++){
            System.out.println(i);
        }
    }
    public void test(){
        execute();     //输出是有序的，说明是同步的
        //execute1();  //输出是无须的，说明是异步的
    }
}
```

线程类：调用不同的方法，于是建立了两个线程类

```
public class ThreadA implements Runnable{
    public void run() {
        TestThread.execute();//调用同步静态方法
    }
}
public class ThreadB implements Runnable{
    public void run() {
```

```
 8        TestThread test=new TestThread();
 9        test.test();//调用非同步非静态方法
10    }
11 }
```

调用：

```
1  Runnable runabbleA=new ThreadA();
2  Thread a=new Thread(runabbleA,"A");
3  a.start();
4  Runnable runabbleB=new ThreadB();
5  Thread b=new Thread(runabbleB,"B");
6  b.start();
```

注意：

1、用synchronized　来锁定一个对象的时候，如果这个对象在锁定代码段中被修改了，则这个锁也就消失了。看下面的实例：

目标类：

```
1 public class TestThread {
2     private static final class TestThreadHolder {
3         private static TestThread theSingleton = new TestThread();
4         public static TestThread getSingleton() {
5             return theSingleton;
6         }
7         private TestThreadHolder() {
8         }
9     }
10
11    private Vector ve =null;
12    private Object lock=new Object();
13    private TestThread(){
14        ve=new Vector();
15        initialize();
16    }
17    public static TestThread getInstance(){
18        return TestThreadHolder.getSingleton();
19    }
20    private void initialize(){
21        for(int i=0;i<100;i++){
22            ve.add(String.valueOf(i));
23        }
```

```
    }
    public void reload(){
        synchronized(lock){
            ve=null;
            ve=new Vector();
                    //lock="abc";
            for(int i=0;i<100;i++){
                ve.add(String.valueOf(i));
            }
        }
        System.out.println("reload end");
    }

    public boolean checkValid(String str){
        synchronized(lock){
            System.out.println(ve.size());
            return ve.contains(str);
        }
    }
}
```

说明：在reload和checkValid方法中都增加了synchronized关键字，对lock对象进行加锁。在不同线程中对同一个对象实例分别调用reload和checkValid方法。
在reload方法中，不修改lock对象即注释lock="abc"; ，结果在控制台输出reload end后才输出100。说明是同步调用的。
如果在reload方法中修改lock对象即去掉注释，结果首先输出了一个数字(当前ve的大小)，然后输出reload end。说明是异步调用的。

2、单例模式中对多线程的考虑

```
public class TestThread {
    private static final class TestThreadHolder {
        private static TestThread theSingleton = new TestThread();
        public static TestThread getSingleton() {
            return theSingleton;
        }
        private TestThreadHolder() {
        }
    }
    private Vector ve =null;
    private Object lock=new Object();
    private TestThread(){
```

```
13        ve=new Vector();
14        initialize();
15    }
16    public static TestThread getInstance(){
17        return TestThreadHolder.getSingleton();
18    }
19        '''
20 }
```

说明：增加了一个内部类，在内部类中申明一个静态的对象，实例化该单例类，初始化的数据都在单例类的构造函数中进行。这样保证了多个实例同时访问的时候，初始化的数据都已经成功初始化了。

总结：

A. 无论synchronized关键字加在方法上还是对象上，它取得的锁都是对象，而不是把一段代码或函数当作锁，所以首先应知道需要加锁的对象

B．每个对象只有一个锁（lock）与之相关联。

C．实现同步是要很大的系统开销作为代价的，甚至可能造成死锁，所以尽量避免无谓的同步控制。

发表于 2007-05-08 11:02 SWINGBOAT 阅读(10670) 评论(15) 编辑 收藏 所属分类: JAVA 、设计模式

评论

re: synchronized的作用

靠 骗人的 怎么运行的结果是一样的

123 评论于 2007-08-16 09:27 回复 更多评论

re: synchronized的作用

@123
呵呵，你把100改成1000试试

456 评论于 2007-08-26 20:16 回复 更多评论

re: synchronized的作用

太谢谢了，受益非浅

wanzhe1945 评论于 2007-08-30 15:55 回复 更多评论

re: synchronized的作用

很好的例子～～～谢谢作者了

thebye85 评论于 2007-11-16 18:33 回复 更多评论

re: synchronized的作用

超级经典的test，受益非浅!

zrwlc2008 评论于 2007-12-11 19:03 回复 更多评论

re: synchronized的作用

我测试的结果为什么是一样的. 都是有顺序的

gaoyuan 评论于 2007-12-18 18:06 回复 更多评论

re: synchronized的作用

那是因为你的循环次数太少了。把100换成1000或更大试试

奔流 评论于 2008-04-01 11:36 回复 更多评论

re: synchronized的作用[未登录]

好！赞一个！

aaa 评论于 2008-04-18 15:38 回复 更多评论

re: synchronized的作用

挺好！！！

1027 评论于 2008-06-01 21:19 回复 更多评论

re: synchronized的作用

谢谢！非常不错～

Dick 评论于 2008-07-05 16:18 回复 更多评论

re: synchronized的作用

讲的很清楚，谢谢

Yvon 评论于 2008-08-20 11:51 回复 更多评论

re: synchronized的作用

谢谢

stp 评论于 2008-09-01 10:49 回复 更多评论

re: synchronized的作用

相当好啊

青叶 评论于 2009-03-26 19:43 回复 更多评论

re: synchronized的作用[未登录]

附上两篇相关的文章:
http://www.ibm.com/developerworks/cn/java/j-jtp10264/index.html
http://www.javaeye.com/topic/48750

swingboat 评论于 2009-05-15 10:06 回复 更多评论

re: synchronized的作用

有意思 呵呵

sheng 评论于 2009-07-08 11:06 回复 更多评论

IT新闻 新用户注册 刷新评论列表

标题 re: synchronized的作用

姓名

主页

验证码 * 2552

内容(请不要发表任何与政治相关的内容)

Remember Me?

提交 登录

[使用Ctrl+Enter键可以直接提交]

该文被作者在 2007-05-08 11:05 编辑过

网站导航:
博客园 个人主页 博客生活 IT博客网 C++博客 博客园社区 管理

相关文章:
利用JGroups同步两台server之间的cache。
有关"+"和"_"的search。

不同时区之间，时间的转换？
maven2使用问题集(命令篇)!
同步（synchronized）对程序性能的影响!
Tomcat5.5.x的JNDI配置（jakarta-tomcat-5.5.7）。
AXIS学习（1）
获取classes目录下的资源文件和类所在目录下的资源文件？
实现类clone方法