

## 设计线程安全的类

- 1:构成对象状态的所有变量
- 2:约束状态变量的不变性条件
- 3:建立对象状态的并发访问管理策略

### 4.1。1收集同步请求

### 4.2实例封闭：

如果某对象不是线程安全的，那么可以通过多种技术使其在多线程程序中安全使用，可以确保该对象只能由单个线程访问，或者通过一个锁来保护对该对象的所有访问

将数据封装在对象内部，可以将数据的访问限制在对象的方法上，从而更容易确保线程在访问数据时总能保持有正确的锁

```
1 package com.test.demo.t4;
2
3 import java.util.HashSet;
4 import java.util.Set;
5
6 public class Test42 {
7
8     public class PersonSet{
9         private final Set<Person> myset = new HashSet<Person>();
10
11         public synchronized void addPerson(Person person){
12             myset.add(person);
13         }
14
15         public synchronized boolean containsPerson(Person p){
16             return myset.contains(p);
17         }
18     }
19     class Person{
20
21     }
22 }
23
```

封闭机制更易于构造线程安全的类

Java监视器模式：

封装对象的所有可变状态，并由对象自己的内置锁来保护

将对象的所有可变状态都进行封装对于任何一种锁对象，只要始终使用该锁对象，都可以用来保护对象的状态

Java监视器模式会将对象的所有可变状态都封装起来，并由对象自己的内置锁来保护

```
1 package com.test.demo.t4;
2
3 import com.sun.xml.internal.xsom.impl.scd.Iterators;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 import java.util.Collections;
9 import java.util.HashMap;
10 import java.util.Map;
11
12 /**
13  * 用Java监视器模式构建车辆追踪器，放宽封装性需求又保持线程安全
14  */
15 @Data
16 public class Test44 {
17
18     private final Map<String,MutablePoint> locations;
19     public Test44(Map<String, MutablePoint> locations){
20
21         this.locations = locations;
22     }
23
24     //调用deepCopy的方法来防止对象发布
25     public synchronized Map<String, MutablePoint> getLocations(){
26         return deepCopy(locations);
27     }
28
29     public synchronized MutablePoint getLocations(String id){
30         return locations.get(id);
31     }
32
33     public synchronized void setLocations(String id,int x,int y){
34         MutablePoint mutablePoint = locations.get(id);
```

```

35     mutablePoint.x = x;
36     mutablePoint.y = y;
37
38 }
39 private static Map<String,MutablePoint> deepCopy(Map<String,
MutablePoint> map){
40     //返回车辆信息的时候,通过 此方法复制正确的值,形成一个新的对象
41     HashMap<String, MutablePoint> map1 = new HashMap<String,
MutablePoint>();
42     for (String s : map1.keySet()) {
43         map1.put(s,new MutablePoint(map.get(s)));
44     }
45
46     //unmodifiableMap 产生一个只读的Map,当你调用此map的put方法时会抛错。
47     Map<String, MutablePoint> map2 = Collections.unmodifiableMap(map1);
48
49
50     return Collections.unmodifiableMap(map1);
51 }
52 //这个内部类是线程不安全的,但上面是线程安全的,他的map和point对象都未发布
53 @Data
54 @AllArgsConstructor
55 static
56 class MutablePoint<T>{
57     public int x;
58     public int y;
59     MutablePoint(MutablePoint p){
60         this.x = p.x;
61         this.y = p.y;
62     }
63     MutablePoint(){
64         x = 0;
65         y = 0;
66     }
67 }
68
69 public static void main(String[] args) {
70
71     void vehicleMoved()
72 }
73 }
74

```

原因：返回客户代码之前复制可变的数据来维持线程安全性，但是在车辆容器非常大的时候极大的降低性能，每次调用getLocation方法都会调用

#### 4.3组合对象

大多数对象是组合对象，将多个非线程安全的类组合成一个线程安全的类时，Java监视器模式是非常有用的

将线程安全委托给多个状态变量，

```
1 package com.test.demo.t4;
2
3 import java.util.List;
4 import java.util.concurrent.CopyOnWriteArrayList;
5
6 /**
7  * 将线程安全委托给多个变量
8  *
9  */
10 public class Test49 {
11     //使用CopyOnWriteArrayList来保存监听器的列表，列表是线程安全的
12     private final List<KeyListener> keyListeners = new
CopyOnWriteArrayList<KeyListener>();
13     //为了将读取的性能发挥到极致，CopyOnWriteArrayList 读取是完全不用加锁的，并且更厉害
的是：写入也不会阻塞读取操作，只有写入和写入之间需要进行同步等待，读操作的性能得到大幅度提
升。
14     private final List<MouseListener> mouseListeners = new
CopyOnWriteArrayList<MouseListener>();
15
16     public void addkeyListeners(KeyListener keyListener){
17         keyListeners.add(keyListener);
18     }
19     public void addMouseListener(MouseListener listener){
20         mouseListeners.add(listener);
21     }
22     public void removeKeyListener(KeyListener listener){
23         keyListeners.remove(listener);
24     }
25     public void removeKeyListener(MouseListener mouseListener){
26         mouseListeners.remove(mouseListener);
27     }
28     class KeyListener{
29
30     }
31     class MouseListener{
```

```
32
33     }
34 }
35
```

使用CopyOnWriteArrayList来保存各个监听列表，是一个线程安全的列表

#### 4.3.3当委托失效的情况

是线程不安全的，没有维持对上界和下界进行约束对不变性

setLower 和 setUpper 都是先检查后执行，没有足够机制保证原子性

```
1 package com.test.demo.t4;
2
3 import java.util.List;
4 import java.util.concurrent.CopyOnWriteArrayList;
5 import java.util.concurrent.atomic.AtomicInteger;
6
7 /**
8  * 将线程安全委托给多个变量
9  *
10 */
11 public class Test410 {
12     final AtomicInteger lower = new AtomicInteger(0);
13     final AtomicInteger upper = new AtomicInteger(0);
14     public static void main(String[] args) {
15
16     }
17
18     public void setLower(int i){
19         //不安全的先检查后执行
20         if (i>upper.get()){
21             throw new IllegalArgumentException();
22         }
23         lower.set(i);
24     }
25
26     public void setUpper(int i){
27         if (i < lower.get()){
28             throw new IllegalArgumentException();
29         }
30         upper.set(i);
31     }

```

```

32
33     public boolean isInRange(int i){
34         return (i>=lower.get() && i<= upper.get());
35     }
36 }
37

```

假设取值范围为 (0, 10) , 如果一个线程调用setLower(5),另一个线程调用setUpper(4),需要避免发布lower和upper

yige 如果一个类是由多个独立且线程安全的状态变量组成, 在所有的操作中都不包含无效状态转变, 那么可以将线程安全性委托给底层的状态变量

#### 4.3.4 发布底层的状态变量

在类中对这些变量施加不变性条件, 才可以发布这些变量。从而使其他类能修改他们

Java



L1 (default)



```

1  package com.test.demo.t4;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6
7  import java.util.Collections;
8  import java.util.Map;
9  import java.util.concurrent.ConcurrentHashMap;
10
11 /**
12  * 将线程安全性托付给底层的ConcurrentHashMap, map中的元素是线程安全的, 是可变的
13  * getlocation返回可变对象的一个不可变副本, 调用者不能增加或者修改车辆位置, 可以通过
14  * 修改map中的Test411的方法值来修改位置 */
15 @Data
16 @AllArgsConstructor
17 @NoArgsConstructor
18 public class Test412 {
19
20     private final Map<String,Test411> locations;
21     private final Map<String,Test411> unmodifiableMap;
22
23     public Test412(Map<String,Test411> locations){
24         this.locations = new ConcurrentHashMap<String, Test411>(locations);
25         this.unmodifiableMap = Collections.unmodifiableMap(this.locations);
26     }

```

```
27
28     public Map<String,Test411> getLocations(){
29         return unmodifiableMap;
30     }
31
32     public Test411 getLocation(String id){
33         return locations.get(id);
34     }
35
36     public void setLocations(String id,int x,int y){
37         locations.get(id).set(x,y);
38     }
39 }
40 class SafePoint{
41
42 }
```