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

```
package com.test.demo.t4;

import java.util.HashSet;
import java.util.Set;

public class Test42 {

    public class PersonSet{
        private final Set<Person> myset = new HashSet<Person>();

        public synchronized void addPerson(Person person){
            myset.add(person);
        }

        public synchronized boolean containsPerson(Person p){
            return myset.contains(p);
        }
    }
    class Person{

    }
}

```

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

```
package com.test.demo.t4;

import com.sun.xml.internal.xsom.impl.scd.Iterators;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

/**
 * 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
 */
@Data
public class Test44 {

    private final Map<String,MutablePoint> locations;
    public Test44(Map<String, MutablePoint> locations){

        this.locations = locations;
    }

    //调用deepCopy的方法来防止对象发布
    public synchronized Map<String, MutablePoint> getLocations(){
        return deepCopy(locations);
    }

    public synchronized MutablePoint getLocations(String id){
        return locations.get(id);
    }

    public synchronized void setLocations(String id,int x,int y){
        MutablePoint mutablePoint = locations.get(id);
```

```
        mutablePoint.x = x;
        mutablePoint.y = y;

    }
    private static Map<String,MutablePoint> deepCopy(Map<String,
MutablePoint> map){
        //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
        HashMap<String, MutablePoint> map1 = new HashMap<String,
MutablePoint>();
        for (String s : map1.keySet()) {
            map1.put(s,new MutablePoint(map.get(s)));
        }

        //unmodifiableMap 产生一个只读的Map，当你调用此map的put方法时会抛错。
        Map<String, MutablePoint> map2 = Collections.unmodifiableMap(map1);


        return Collections.unmodifiableMap(map1);
    }
    //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
    @Data
    @AllArgsConstructor
    static
    class MutablePoint<T>{
        public int x;
        public int y;
        MutablePoint(MutablePoint p){
        this.x = p.x;
        this.y = p.y;
        }
        MutablePoint(){
            x = 0;
            y = 0;
        }
    }

    public static void main(String[] args) {

        void vehicleMoved()
    }
}

```