一: 不可变对象设计模式

其实这种模式的本质就是每次的添加修改数据的时候，创建一个新的对象，那样就可以保证不会出现资源的竞争问题了，因此这种方式在String里面是非常适合使用的。下面就先列举一个小例子，是可变对象的例子，做一个小的累加器。

```
package immutable;

import java.util.concurrent.TimeUnit;
import java.util.stream.IntStream;

public class IntegerAccumulatorMutable {
    private  int init;

    //构造时传入初始值
    public IntegerAccumulatorMutable(int init) { this.init = init; }


    public int add(int i) {
        this.init += i;
        return this.init;
    }

    public int getValue() { return this.init; }

    public static void main(String[] args) {
        IntegerAccumulatorMutable accumulator = new IntegerAccumulatorMutable(0);
        System.out.println("默认初始化的对象地址为: "+accumulator);
        IntStream.range(0, 3).forEach(i -> new Thread(
                  () -> {
                      int inc = 0;
                      while(inc <100) {
                          int oldValue = accumulator.getValue();
                          //System.out.println("oldValue的对象的地址为: "+accumulator);
                          int result = accumulator.add(inc);
                          System.out.println(oldValue + "+" + inc +"="+result);
                          if(inc + oldValue != result) {
                              System.out.println("ERROR: "+oldValue+ " + "+inc+" = "+result);
                          }
                          inc++;
                          slowly();
                      }
                  }).start());

    }

    private static void slowly() {
        try {
            TimeUnit.MILLISECONDS.sleep(1);
        }catch(Exception e) {
            e.printStackTrace();
        }
    }

}
```

这段代码很简单，然后运行结果如下所示:

```
...
13490+95=13585
13680+96=13776
13872+96=13968
13776+96=13872
13968+97=14065
14162+97=14259
14065+97=14162
14259+98=14357
14455+98=14553
14259+98=14455
ERROR: 14259 + 98 = 14455
14553+99=14652
14652+99=14751
14751+99=14850
...
```

这里就直观地看到错误的地方和原因了，那么下面就修改一下代码，成为一个不可变对象的设计方式

```
 package immutable;
```

```
import java.util.concurrent.TimeUnit;
import java.util.stream.IntStream;;

/**
 * 这里是设计一个不可变对象的代码，本质就是每次都是使用一个新的对象空间
 * 采用final是不允许继承，防止修改
 * @author hetao
 *
 */
public final class IntegerAccumulator {

    private final int init;

    //构造时传入初始值
    public IntegerAccumulator(int init) {
        this.init = init;
    }

    public IntegerAccumulator(IntegerAccumulator accumulator,int init) {
        this.init = accumulator.getValue()+init;
    }

    //每次相加都会产生一个新的对象
    public IntegerAccumulator add(int i) {
        IntegerAccumulator accumulator = new IntegerAccumulator(this,i);
        System.out.println("当前的对象地址为: "+accumulator);
        return accumulator;
    }

    public int getValue() {
        return this.init;
    }

    public static void main(String[] args) {
        IntegerAccumulator accumulator = new IntegerAccumulator(0);
        System.out.println("默认初始化的对象地址为: "+accumulator);
        IntStream.range(0, 3).forEach(i -> new Thread(
                  () -> {
                      int inc = 0;
                      while(inc <10) {
                          int oldValue = accumulator.getValue();
                          System.out.println("oldValue的对象的地址为: "+accumulator);
                          int result = accumulator.add(inc).getValue();
                          System.out.println(oldValue + "+" + inc +"="+result);
                          if(inc + oldValue != result) {
                              System.out.println("ERROR: "+oldValue+ " + "+inc+" = "+result);
                          }
                          inc++;
                          slowly();
                      }
                  }).start());

    }

    private static void slowly() {
        try {
            TimeUnit.MILLISECONDS.sleep(1);
        }catch(Exception e) {
            e.printStackTrace();
        }
    }

}
```

这里的修改关键地方在add方法这里，每次add累加的时候，都是会直接new一个构造方法，那么就必然会产生了一个新的对象了，但是这样也会导致一个新的问题出现，看结果即可知道。

```
默认初始化的对象地址为: immutable.IntegerAccumulator@3d4eac69
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
当前的对象地址为: immutable.IntegerAccumulator@2e22a55b
0+0=0
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
当前的对象地址为: immutable.IntegerAccumulator@4229a8e7
当前的对象地址为: immutable.IntegerAccumulator@5c989a83
0+0=0
0+0=0
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
当前的对象地址为: immutable.IntegerAccumulator@525f6f60
```

```
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
当前的对象地址为: immutable.IntegerAccumulator@4cee5ed2
0+1=1
0+1=1
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
当前的对象地址为: immutable.IntegerAccumulator@20e54eef
0+1=1
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
当前的对象地址为: immutable.IntegerAccumulator@48d1c397
0+2=2
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
当前的对象地址为: immutable.IntegerAccumulator@5660f1f3
oldValue的对象的地址为: immutable.IntegerAccumulator@3d4eac69
当前的对象地址为: immutable.IntegerAccumulator@24dace2b
0+2=2
0+2=2
...
```

这里可以看到oldValue的值是没有改变的，这个原因就是因为result的结果是一个新的对象，但是oldValue的还是原来旧的对象，所以值并没有更新。所以一般这种设计比较适合于String字符串这些的，因为不用考虑修改后的对象的值的问题。