> java中实现观察者模式有很多种方式，上一篇文章介绍到了，利用callback的方式实现了回调，这篇文章准备介绍的是利用listener实现回调。

Java回调机制

根据实时性划分：

- 同步回调
- 异步回调

实现方式

- 利用匿名内部类即callbck来实现
- 用listener来实现

这两种实现方式本质上是类似的，应用场景略有不同，如果有熟知安卓的朋友应该可以知道，在为一个view添加点击实现的时候是有两种方式的

1. 利用callback来实现

```
view.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View view) {

            }
        });
```

2.实现View.OnClickListener接口

```
public class Test extends AppCompatActivity implements View.OnClickListener {
    private Button button;

    @Override protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        button = (Button) findViewById(R.id.ac_video_btn_getValue);
        button.setOnClickListener(this);
    }

    @Override public void onClick(View view) {
        //TODO
    }
}
```

回调的本质

其实无论哪种方式来实现回调，利用的思想都是观察者模式，即在我们选择订阅之后，当对方做出任何举动的时候会给我们发送一条信息，这样做的好处是省着我们用一个新的线程轮训检测对方的状态，可以节省很多的资源。

应用的场景

- 如果我们需要将信息一层一层的返回去的时候，正如我下面的例子，那么可能用listener更为适合我们，因为我们可以将这个listener进行传递，在需要查看数据的时候进行回调它。或者当我们有很多事件需要回调的时候，可以实现一个listener然后发送不同的信息，进行区分。这样代码看起来会简洁一些，不会像callback一样，会嵌套很多层，也不会写出很多个callback来。
- 如果我们只有一处，或者简单的几处需要回调的话，那么我们完全可以不用实现这个接口，而是用callback的方式来进行处理。

还是举一个简单的生活中的例子吧，在公司中有一件事情，老板想要问员工，但是老板只能联系到部门经理，那么便有了，A问B,B问C，C经过思考，回答了B，B又将答案告诉了A，A知道了答案，便高兴的说了出来。

我下面的代码采用了单例模式来写：

Listner.java

```
public interface Listener {
    void onFinish(String msg);
}
```

People.java

```
public class People implements Listener {

    private static People people;

    private People() {

    }

    synchronized public static People getInstance() {
        if (people == null) {
            people = new People();
        }
        return people;
    }

    public void askPeople2() {
        People2.getInstance().askPeople3(this);
    }

    @Override
    public void onFinish(String msg) {
        System.out.println("收到的消息是 ---> " + msg);
    }

}
```

People2.java

```
public class People2 {

    private static People2 people2;

    private People2() {

    }

    synchronized public static People2 getInstance() {
        if (people2 == null)
            people2 = new People2();
        return people2;
    }

    public void askPeople3(Listener listener){
        People3.getInstance().thinking(listener);
    }

}
```

People3.java

```
public class People3 {

    private static People3 people3;

    private People3() {

    }

    synchronized public static People3 getInstance() {
        if (people3 == null)
            people3 = new People3();
        return people3;
    }

    public void thinking(Listener listener3) {
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        listener3.onFinish("我已经思考完毕");
    }

}
```