链滴

一个简单的观察者模式

使用JDK自带的 观察者模式

被观察者类，需要继承Observable类

```
import java.util.Observable;

/**
 * Created by sean on 2017/11/23 9:28.
 * 这是一个继承了Observable的观察者模式，改类为被观察者
 */
public class LoginWithJDK extends Observable {

    public static void main(String[] args) {
        LoginWithJDK loginWithJDK = new LoginWithJDK();
        ObserverWithJDK(loginWithJDK);

        loginWithJDK.loginSuccess();

    }

    public void loginSuccess(){
        System.out.println("登录成功，通知观察者...");

        //下面两个为父类的方法
        setChanged();
        notifyObservers();
    }
}
```

观察者类，需要先实现Observer接口，并重写update()方法

```
import java.util.Observable;
import java.util.Observer;

/**
 * Created by sean on 2017/11/23 9:31.
 * 改类为观察者类，当被观察者发生变化时调用update方法
 */
public class ObserverWithJDK implements Observer{

    public ObserverWithJDK(LoginWithJDK loginWithJDK){
        loginWithJDK.addObserver(this);
    }

    @Override
    public void update(Observable o, Object arg) {
        System.out.println("被观察者的方法被触发");
    }
}
```

接口Observer的源码

```
package java.util;
```

```
/**
 * A class can implement the <code>Observer</code> interface when it
 * wants to be informed of changes in observable objects.
 *
 * @author  Chris Warth
 * @see     java.util.Observable
 * @since   JDK1.0
 */
public interface Observer {
    void update(Observable o, Object arg);
}

package java.util;

public class Observable {
    private boolean changed = false;
    private Vector<Observer> obs;

    public Observable() {
        obs = new Vector<>();
    }

    public synchronized void addObserver(Observer o) {
        if (o == null)
            throw new NullPointerException();
        if (!obs.contains(o)) {
            obs.addElement(o);
        }
    }

    public synchronized void deleteObserver(Observer o) {
        obs.removeElement(o);
    }

    public void notifyObservers() {
        notifyObservers(null);
    }

    public void notifyObservers(Object arg) {
        Object[] arrLocal;

        synchronized (this) {

            if (!changed)
                return;
            arrLocal = obs.toArray();
            clearChanged();
        }

        for (int i = arrLocal.length-1; i>=0; i--)
            ((Observer)arrLocal[i]).update(this, arg);
    }
    public synchronized void deleteObservers() {
        obs.removeAllElements();
    }

    protected synchronized void setChanged() {
        changed = true;
    }

    protected synchronized void clearChanged() {
        changed = false;
    }

    public synchronized boolean hasChanged() {
        return changed;
    }

    public synchronized int countObservers() {
        return obs.size();
    }
}
```

不适用JDK中方法创建一个观察者模式案例

观察者需要实现的接口

```
public interface ObserverInterface {

    void onSuccess(Object object);
}
```

观察者，需要实现上面接口

```
public class ObserverOriginal implements ObserverInterface{

    public ObserverOriginal(LoginOriginal loginOriginal){
        loginOriginal.addObject(this);
    }

    @Override
    public void onSuccess(Object object) {
        System.out.println("登录成功，通知观察者...");
    }
}
```

被观察者，一个登陆的类，登陆成功之后通知观察者

```
public class LoginOriginal{

    private List<ObserverInterface> loginInterfaces = new ArrayList<>();

    public void addObject(ObserverInterface addObject){
        loginInterfaces.add(addObject);
    }

    private void loginSuccess(){
        for (ObserverInterface loginInterface : loginInterfaces) {
            loginInterface.onSuccess(this);
        }
    }

    public static void main(String[] args) {
        LoginOriginal loginOriginal = new LoginOriginal();
        new ObserverOriginal(loginOriginal);
        loginOriginal.loginSuccess();
    }
}
```

原文链接：一个简单的观察者模式