AlexOne

二、观察者模式

目录

概述：

定义一个一对多的依赖关系，使得每当一个对象改变状态，则所有依赖它的对象都会得到通知并被自动更新，又叫订阅模式。
北京时间一动，其它的表跟着动。该例子是传统方法实现的一个动，其它的跟着动。

传统方法的示例：

```
#include <iostream>

using namespace std;

class B{
public:
    void update(int x){
        _x=x;
        _x+=10;
        cout<<"B:"<<_x<<endl;
    }
    int _x;
};

class C{
public:
    void update(int x){
        _x=x;
        _x+=20;
        cout<<"C:"<<_x<<endl;
    }
    int _x;
};

class D{
public:
    void update(int x){
        _x=x;
        _x+=30;
        cout<<"D:"<<_x<<endl;
```

```
    }
    int _x;
};

class A{
public:
    void update(B *pb,C *pc,D *pd,int x){
        _x=x;
        pb->update(_x);
        pc->update(_x);
        pd->update(_x);
    }
    int _x;
};

int main(){
    B b; C c; D d;
    A a;
    a.update(&b,&c,&d,20);
    return 0;
}
```

观察者模式的示例：

```
#include <iostream>
#include <list>
#include <algorithm>

using namespace std;

// 相比传统方法，可以取消订阅者，比较灵活。符合：对扩展开放，对修改关闭。
// 关键之处：list<Observer *>

class Observer{
public:
    virtual void update(int hour,int min,int sec)=0;
};

class Subject{                          //这些虚函数在PekingTimeSubject里面覆写
public:
    virtual void registerObserver(Observer *ob)=0;
    virtual void removeObserver(Observer *ob)=0;
    virtual void notify()=0;
protected:
    list<Observer *> observerList;       // 观察者列表
};

class PekingTimeSubject:public Subject{
public:
    void setTimer(int hour,int min,int sec){
        _hour=hour;
        _min=min;
        _sec=sec;
        notify();
    }
    void registerObserver(Observer *ob){
        observerList.push_back(ob);
    }
```

```
    void removeObserver(Observer *ob){
        observerList.remove(ob);         // lsit的remove方法
    }
    void notify(){                       // 此函数必须在子类中实现，因为要接触数据。
        list<Observer *> :: iterator itr =observerList.begin();
        for(;itr != observerList.end();itr++){
            (*itr)->update(_hour,_min,_sec);
        }                                // 遍历list去进行挨个通知
    }
private:
    int _hour;
    int _min;
    int _sec;
};

class AmericaTimerObserver:public Observer{
public:
    void update(int hour,int min,int sec){    //覆写update方法
        _hour=hour;
        _min=min;
        _sec=sec;
        dis();
    }
    void dis(){
        cout<<"America Time is update"<<endl;
        cout<<"H:"<<_hour<<" M:"<<_min<<" S:"<<_sec<<endl;
    }
private:
    int _hour;
    int _min;
    int _sec;
};

class JapanTimerObserver:public Observer{
public:
    void update(int hour,int min,int sec){    //覆写update
        _hour=hour;
        _min=min;
        _sec=sec;
        dis();
    }
    void dis(){
        cout<<"Japan Time is update"<<endl;
        cout<<"H:"<<_hour<<" M:"<<_min<<" S:"<<_sec<<endl;
    }
private:
    int _hour;
    int _min;
    int _sec;
};

int main(){
    PekingTimeSubject *bj = new PekingTimeSubject;        // 北京时间
    JapanTimerObserver *jp = new JapanTimerObserver;      // 东京时间
    AmericaTimerObserver *am = new AmericaTimerObserver;  // 美国时间
    bj->registerObserver(jp);     // bj就是被观察者，要把观察者都注册进去
    bj->registerObserver(am);     // 注意这里使用北京时间来作为联动flag

    bj->setTimer(10,20,30);       // 同时设置东京时间和美国时间
    bj->removeObserver(jp);       // 移除东京时间
    bj->setTimer(1,2,3);          // 此时的时间设置只会对美国时间有效
```

```
    // 继承subject的才可以register, jp、am不可register。
    return 0;
}
```