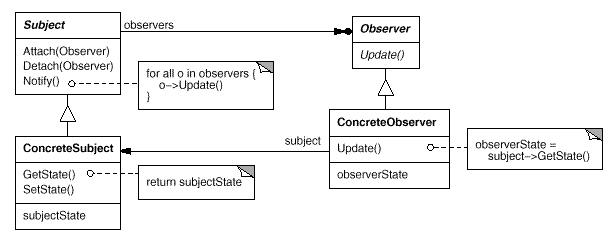
Observer



Observer.h

#ifndef \_OBSERVER\_H

#define \_OBSERVER\_H

#include "Subject.h"

using namespace *std*;

class Subject;

class Observer

{

public:

Observer() {

m\_nObserverState = -1;

};

virtual ~Observer() {};

//纯虚函数 基类不实现 派生类必须实现

virtual void UpDate(Subject\* **pSubject**) = 0;//通知 Observer 状态发生了变化

protected:

int m\_nObserverState; //保存Observe状态

};

class ConcreteObserver :public Observer

{

public:

ConcreteObserver(){}

virtual ~ConcreteObserver() {}

void UpDate(Subject\* **pSubject**);

};

#endif

Observer.cpp

#include "Observer.h"

void ConcreteObserver::UpDate(Subject\* **pSubject**)

{

if (*NULL* != **pSubject**) {

m\_nObserverState = **pSubject**->GetState();

*std*::*cout* << "The ConcreteObserver is "<< m\_nObserverState <<*std*::*endl*;

}

}

Subject.h

#ifndef \_SUBJECT\_H

#define \_SUBJECT\_H

#include <list>

#include <iostream>

#include "Observer.h"

using namespace *std*;

class Observer;

class Subject

{

public:

Subject() {

m\_nSubjectState = -1;

};

virtual ~Subject();

void Attach(Observer\* **pObserver**); //新增对象

void Detach(Observer\* **pObserver**); //删除对象

void Notify(); //通知对象改变状态

// 虚函数，提供默认的实现，派生类可以自己实现来覆盖 基类的实现

//设置状态

virtual void SetState(int **nState**);

//获取状态

virtual int GetState();

protected:

int m\_nSubjectState; //模拟保存Subject状态变量

private:

*std*::*list*<Observer\*> m\_ListObserver;//保存Observer指针链表

};

class ConcreteSubject :public Subject

{

public:

ConcreteSubject() {};

virtual ~ConcreteSubject() {};

// 派生类自己实现来覆盖 基类的实现

virtual void SetState(int **nState**); //设置状态

virtual int GetState(); //获取状态

};

#endif

Subject.cpp

#include "Subject.h"

Subject::~Subject()

{

*std*::*list*<Observer\*>::*iterator* **iter**;

for (**iter** = m\_ListObserver.*begin*(); **iter** != m\_ListObserver.*end*(); **iter**++)

{

delete(\***iter**);

}

m\_ListObserver.*clear*();

*std*::*cout* << "m\_ListObserver.clear() \n" << *std*::*endl*;

}

void Subject::Attach(Observer\* **pObserver**)

{

if (*NULL* != **pObserver**){

m\_ListObserver.*push\_back*(**pObserver**);

}

*std*::*cout* << "Attach an Observer \n" << *std*::*endl*;

}

void Subject::Detach(Observer\* **pObserver**)

{

*std*::*list*<Observer\*>::*iterator* **iter**;

if (*NULL* != **pObserver**){

**iter** = *std*::*find*(m\_ListObserver.*begin*(),m\_ListObserver.*end*(),**pObserver**);

if (**iter** != m\_ListObserver.*end*())

{

m\_ListObserver.*erase*(**iter**);

}

}

*std*::*cout* << "Detach an Observer \n" << *std*::*endl*;

}

void Subject::Notify()

{

*std*::*cout* << "Notify ObserverS State \n" << *std*::*endl*;

*std*::*list*<Observer\*>::*iterator* **iter**;

for (**iter** = m\_ListObserver.*begin*();**iter** != m\_ListObserver.*end*();**iter**++)

{

(\***iter**)->UpDate(this);

}

}

void Subject::SetState(int **nState**)

{

*std*::*cout* << "State Set By Subject \n" << *std*::*endl*;

m\_nSubjectState = **nState**;

}

int Subject::GetState()

{

*std*::*cout* << "State Get By Subject \n" << *std*::*endl*;

return m\_nSubjectState;

}

/\*

ConcreteSubject 类成员函数的实现

\*/

void ConcreteSubject::SetState(int **nState**)

{

*std*::*cout* << "SetState By ConcreteSubject \n" << *std*::*endl*;

m\_nSubjectState = **nState**;

}

int ConcreteSubject::GetState()

{

*std*::*cout* << "Get By ConcreteSubject \n" << *std*::*endl*;

return m\_nSubjectState;

}

main.cpp

#include <stdio.h>

#include "Observer.h"

int main()

{

*printf*("Observer Test \n");

Observer\* **pObserver1** = new ConcreteObserver;

Observer\* **pObserver2** = new ConcreteObserver;

Subject\* **pSbuject** = new Subject;

**pSbuject**->Attach(**pObserver1**); //新增对象

**pSbuject**->Attach(**pObserver2**);

**pSbuject**->SetState(true); //设置状态

**pSbuject**->Notify(); //通知对象改变状态 Observer->UpDate() pSubject->GetState();

**pSbuject**->Detach(**pObserver2**); //删除对象

**pSbuject**->SetState(false); //设置状态

**pSbuject**->Notify(); //通知对象改变状态 Observer->UpDate() pSubject->GetState();

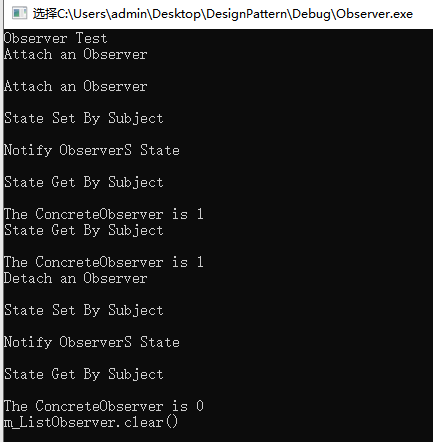
delete **pSbuject**;

*system*("pause\n");

}



输出结果



Adapter

