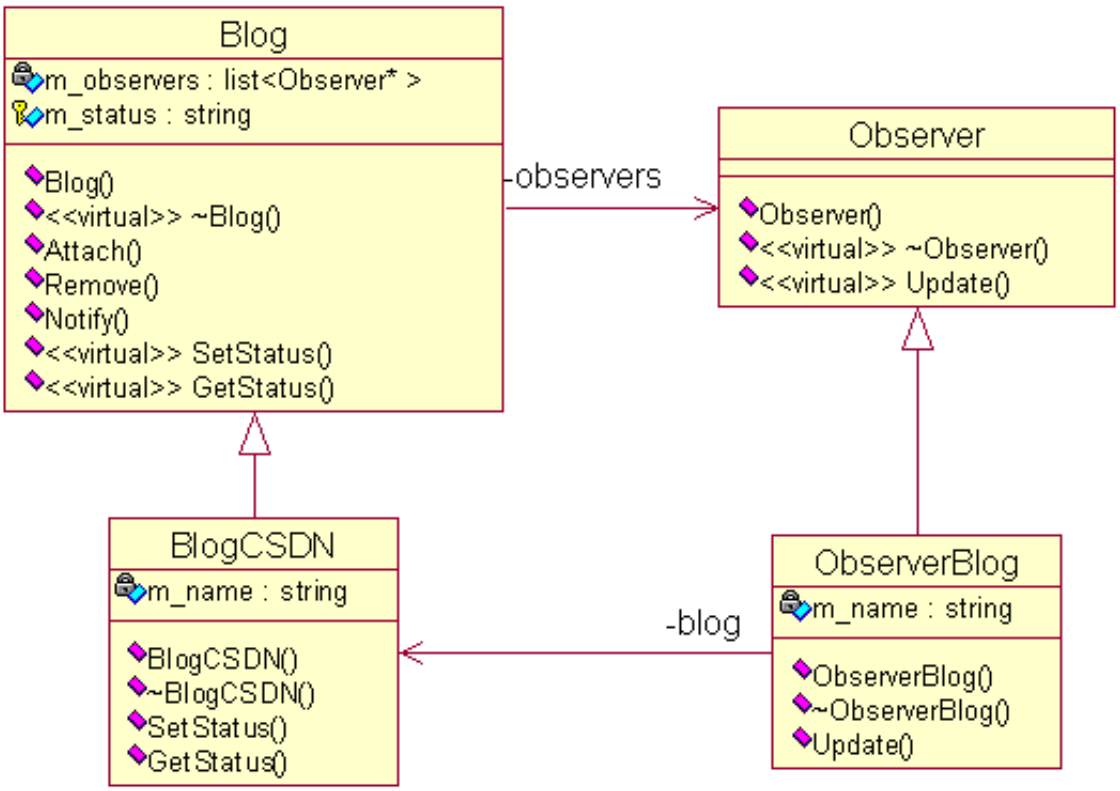
設計模式C++實現（15）——觀察者模式

星期六, 2013 12月 14, 1:13 上午

軟件領域中的設計模式為開發人員提供了一種使用專家設計經驗的有效途徑。設計模式中運用了面向對象編程語言的重要特性：封裝、繼承、多態，真正領悟設計模式的精髓是可能一個漫長的過程，需要大量實踐經驗的積累。最近看設計模式的書，對於每個模式，用C++寫了個小例子，加深一下理解。主要參考《大話設計模式》和《設計模式:可復用面向對象軟件的基礎》兩本書。本文介紹觀察者模式的實現。

觀察者模式：定義對象間的一種一對多的依賴關係，當一個對象的狀態發生改變時，所有依賴於它的對象都得到通知並被自動更新。它還有兩個別名，依賴(Dependents)，發佈-訂閱(Publish-Subsrcibe)。可以舉個博客訂閱的例子，當博主發表新文章的時候，即博主狀態發生了改變，那些訂閱的讀者就會收到通知，然後進行相應的動作，比如去看文章，或者收藏起來。博主與讀者之間存在種一對多的依賴關係。下面給出相應的UML圖設計。



可以看到博客類中有一個觀察者鏈表（即訂閱者），當博客的狀態發生變化時，通過Notify成員函數通知所有的觀察者，告訴他們博客的狀態更新了。而觀察者通過Update成員函數獲取博客的狀態信息。代碼實現不難，下面給出C++的一種實現。

```
//觀察者
class Observer
{
public:
    Observer() {}
    virtual ~Observer() {}
    virtual void Update() {}
};
//博客
```

```
10. class Blog
11. {
12. public:
13.     Blog() {}
14.     virtual ~Blog() {}
15.     void Attach(Observer *observer) { m_observers.push_back(observer); }     //添加觀察者
16.     void Remove(Observer *observer) { m_observers.remove(observer); }        //移除觀察者
17.     void Notify() //通知觀察者
18.     {
19.         list<Observer*>::iterator iter = m_observers.begin();
20.         for(; iter != m_observers.end(); iter++)
21.             (*iter)->Update();
22.     }
23.     virtual void SetStatus(string s) { m_status = s; } //設置狀態
24.     virtual string GetStatus() { return m_status; }    //獲得狀態
25. private:
26.     list<Observer* > m_observers; //觀察者鏈表
27. protected:
28.     string m_status; //狀態
29. };
```

以上是觀察者和博客的基類，定義了通用接口。博客類主要完成觀察者的添加、移除、通知操作，設置和獲得狀態僅僅是一個默認實現。下面給出它們相應的子類實現。

```
1.  //具體博客類
2.  class BlogCSDN : public Blog
3.  {
4.  private:
5.      string m_name; //博主名稱
6.  public:
7.      BlogCSDN(string name): m_name(name) {}
8.      ~BlogCSDN() {}
9.      void SetStatus(string s) { m_status = "CSDN通知：" + m_name + s; } //具體設置狀態信息
10.     string GetStatus() { return m_status; }
11. };
12. //具體觀察者
13. class ObserverBlog : public Observer
14. {
15. private:
16.     string m_name;  //觀察者名稱
17.     Blog *m_blog;   //觀察的博客，當然以鏈表形式更好，就可以觀察多個博客
18. public:
19.     ObserverBlog(string name,Blog *blog): m_name(name), m_blog(blog) {}
20.     ~ObserverBlog() {}
21.     void Update()  //獲得更新狀態
22.     {
23.         string status = m_blog->GetStatus();
24.         cout<<m_name<<"-------"<<status<<endl;
25.     }
26. };
```

客戶的使用方式：

```
1. //測試案例
2. int main()
3. {
4.     Blog *blog = new BlogCSDN("wuzhekai1985");
5.     Observer *observer1 = new ObserverBlog("tutupig", blog);
6.     blog->Attach(observer1);
7.     blog->SetStatus("發表設計模式C++實現（15）——觀察者模式");
8.     blog->Notify();
9.     delete blog; delete observer1;
10.    return 0;
11. }
```