观察者模式 - 摩根斯

发表于 2022-09-18 | 更新于 2023-04-06 | 设计模式

| 字数总计: 2.3k | 阅读时长: 7 分钟 | 阅读量: 1034 | 评论数: 2

配套视频课程已更新完毕，大家可通过以下两种方式观看视频讲解：

关注公众号： 爱编程的大丙 ，或者进入 大丙课堂 学习。

苏丙榅

合抱之木，生于毫末；九层之台，起于垒土；千里之行，始于足下。

1. 新闻大亨

摩根斯是海贼世界中一个比较神秘的人物，他是世界经济新闻社的社长，人称 “大新闻摩根斯”。他总是能非常轻松地搞到第一手情报，将其印成报纸，并由自家的送报鸟把报纸送到世界各地。

先不去深究摩根斯是如何快速得到世界各地的动态以及得到一些秘密情报，暂且认为他是豢养了一支空中狗仔队，在全世界的上空进行 24 小时无死角监视。对于摩根斯的新闻社我们可以将其看作是消息的发布者，对于购买报纸的各国人民或者是海上的海贼，我们可以将他们看作是消息的观察者或者订阅者。

在设计模式中也有一种类似的描述行为的模式，叫做观察者模式。观察者模式允许我们定义一种订阅机制，可在对象事件发生时通知所有的观察者对象，使它们能够自动更新。观察者模式还有另外一个名字叫做 “发布 - 订阅” 模式。

目录

观察者模式在日常生活中也很常见，比如：

- 使用的社交软件，当关注的博主更新了内容，会收到提示信息
- 购买的商品被送到菜鸟驿站，会收到驿站发送的提示信息
- 订阅了报刊，每天 / 每月都会收到新的报纸或者杂志

2. 辛苦了，送报鸟

2.1 发布者

摩根斯的新闻社是一个报纸发行机构，内容大多是与当前的世界格局和各地发生的事件有关，其他人也可以发报纸，为了避免竞争可以更换一下题材，比如海贼们的八卦新闻，不管是哪一类都需要满足以下的需求：

1. 添加订阅者，将所有的订阅者存储起来
2. 删除订阅者，将其从订阅者列表中删除
3. 将消息发送给订阅者（发通知）

根据上述信息，我们就可以先创建出一个发布者的抽象类：

头文件 NewsAgency.h

C++

```
// 声明订阅者类，只是做了声明，并没有包含这个类的头文件
class Observer;
```

CMake 保姆级教程（下）
2023-03-15

CMake 保姆级教程（上）
2023-03-06

访问者模式 - 再见，香波地群岛
2022-09-22

模板方法模式 - 和平主义者
2022-09-21

状态模式 - 文斯莫克・山治
2022-09-20

```
3  // 新闻社
4  class NewsAgency
5  {
6  public:
7      void attach(Observer* ob);
8      void deatch(Observer* ob);
9      virtual void notify(string msg) = 0;
10     virtual ~NewsAgency() {};
11 protected:
12     // 订阅者列表
13     list<Observer*> m_list;
14 };
```

对于上面定义的类做以下的细节说明:

- 第 2 行： `class Observer` ，此时这个 `Observer 订阅者类` 还不存在，此处只是做一个声明，让编译器能够通过编译
- 第 13 行：将所有的订阅者对象存储到 STL 的 `list 容器中`
- 第 7 行：添加一个订阅者到 `list 容器中`
- 第 8 行：从 `list 容器中` 删除一个订阅者
- 第 9 行：将通知信息发送给 `list 容器中` 的所有订阅者

源文件 NewsAgency.cpp

C++

```
1  #include "NewsAgency.h"
2  #include "Observer.h"    // 在源文件中包含该头文件
```

```cpp
#include <iostream>
void NewsAgency::attach(Observer* ob)
{
    m_list.push_back(ob);
}

void NewsAgency::deatch(Observer* ob)
{
    m_list.remove(ob);
}
```

> 在头文件中只是对 Observer 类进行了声明定义了这种类型的指针变量，在源文件中需要调用 Observer 类提供的 API，所以必须要包含这个类的头文件。这么处理的目的是为了避免两个相互关联的类他们的头文件相互包含。

摩根斯新闻

摩根斯的新闻社可以作为上面 `NewsAgency 类` 的子类，在子类中重写父类的纯虚函数 `notify()` 方法就可以了。

头文件 NewsAgency.h

C++

```cpp
// 摩根斯的新闻社
class Morgans : public NewsAgency
{
public:
```

```
    void notify(string msg) override;
};
```

源文件 NewsAgency.cpp

C++

```
void Morgans::notify(string msg)
{
    cout << "摩根斯新闻社报纸的订阅者一共有<" << m_list.size() << ">人" << endl;
    for (const auto& item : m_list)
    {
        item->update(msg);      // 订阅者类的定义在下面
    }
}
```

八卦新闻

八卦新闻社的处理思路和摩根斯新闻社的处理思路是完全一样的，代码如下：

头文件 NewsAgency.h

C++

```
// 八卦新闻
class Gossip : public NewsAgency
{
public:
    void notify(string msg) override;
};
```

源文件 NewsAgency.cpp

C++

```cpp
void Gossip::notify(string msg)
{
    cout << "八卦新闻社报纸的订阅者一共有<" << m_list.size() << ">人" << endl;
    for (const auto& item : m_list)
    {
        item→update(msg);
    }
}
```

2.2 订阅者

虽然在海贼王中尾田构建的是一个强者的世界，但是还是有温情存在的，路飞虽然不知道自己的老爹长啥样、是干什么的，但是知道自己还有个儿子的龙还是一直在默默关注着路飞。另外，还有把未来希望寄托在路飞身上的香克斯，也一直在关注着路飞的成长。所以观察者这个角色可能不是一个人，可能是几个或者是一个群体，但他们的行为是一致的，所以我们可以给所有的观察者定义一个抽象的基类。

C++

```cpp
#pragma once
#include <string>
#include <iostream>
#include "NewsAgency.h"
using namespace std;

```

```
7  // 抽象的订阅者类
8  class Observer
9  {
10 public:
11     Observer(string name, NewsAgency* news) :m_name(name), m_news(news)
12     {
13         m_news->attach(this);
14     }
15     void unsubscribe()
16     {
17         m_news->deatch(this);
18     }
19     virtual void update(string msg) = 0;
20     virtual ~Observer() {}
21 protected:
22     string m_name;
23     NewsAgency* m_news;
24 };
```

下面介绍一下这个抽象的观察者类中的一些细节：

- 第 11 行：需要通过构造函数给观察者类提供一个信息的发布者
- 第 13 行：通过发布者对象将观察者对象存储了起来，这样就可以收到发布者推送的消息了。
- 第 15 行：观察者取消订阅，取消之后将不再接收订阅消息。
- 第 19 行：观察者得到最新消息之后，用于更新自己当前的状态。

有了上面的抽象类，我们就可以再添加几个订阅者的子类：

蒙奇·D·龙

C++

```cpp
class Dragon : public Observer
{
public:
    using Observer::Observer;
    void update(string msg) override
    {
        cout << "@@@路飞的老爸革命军龙收到消息: " << msg << endl;
    }
};
```

香克斯

C++

```cpp
class Shanks : public Observer
{
public:
    using Observer::Observer;
    void update(string msg) override
    {
        cout << "@@@路飞的引路人红发香克斯收到消息: " << msg << endl;
    }
};
```

巴托洛米奥

C++

```cpp
class Bartolomeo : public Observer
{
public:
    using Observer::Observer;
    void update(string msg) override
    {
        cout << "@@@路飞的头号粉丝巴托洛米奥收到消息: " << msg << endl;
    }
};
```

可以看到上面的三个子类非常类似，只是在各自的类中分别重写了 `update()` 这个状态更新函数（因为以上是测试程序，所有逻辑比较简单）。

2.3 起飞，送报鸟

最后，我们来演示一下消息从发布到到达订阅者手中的过程，在此要感谢送报鸟的辛勤付出：

C++

```cpp
int main()
{
    Morgans* ms = new Morgans;
    Gossip* gossip = new Gossip;
    Dragon* dragon = new Dragon("蒙奇·D·龙", ms);
    Shanks* shanks = new Shanks("香克斯", ms);
    Bartolomeo* barto = new Bartolomeo("巴托洛米奥", gossip);
    ms→notify("蒙奇·D·路飞成为新世界的新的四皇之一，赏金30亿贝里!!!");
    cout << "=====================================" << endl;
    gossip→notify("女帝汉库克想要嫁给路飞，给路飞生猴子，哈哈哈...");

```

```
    delete ms;
    delete gossip;
    delete dragon;
    delete shanks;
    delete barto;

    return 0;
}
```

送报鸟送到订阅者手中的信息是这样的:

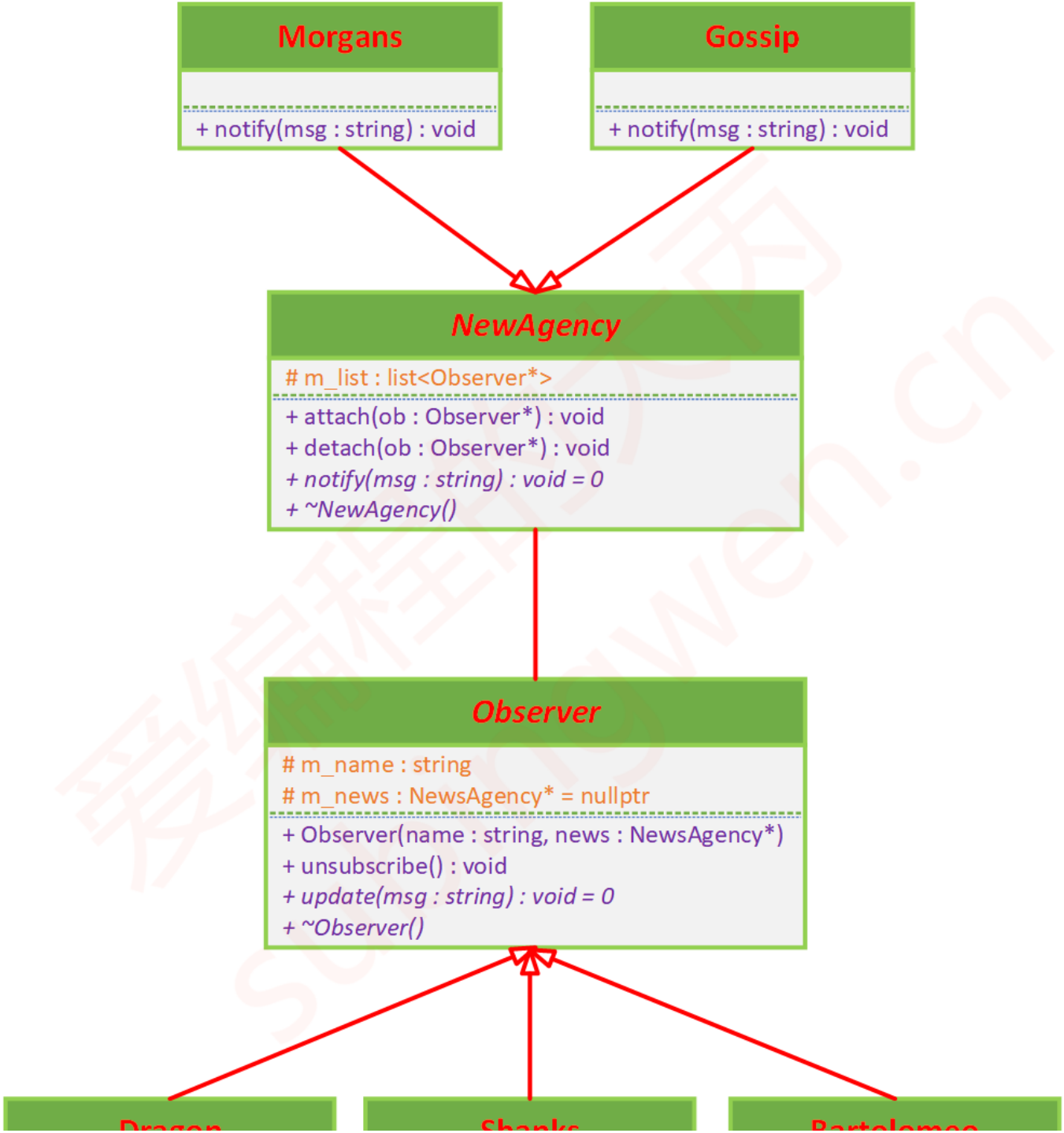
C++

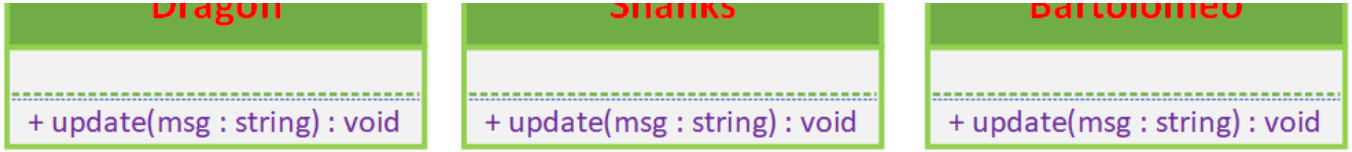
```
摩根斯新闻社报纸的订阅者一共有<2>人
@@@路飞的老爸革命军龙收到消息： 蒙奇·D·路飞成为新世界的新的四皇之一， 赏金30亿贝里!!!
@@@路飞的引路人红发香克斯收到消息： 蒙奇·D·路飞成为新世界的新的四皇之一， 赏金30亿贝里!!!
=====================================
八卦新闻社报纸的订阅者一共有<1>人
@@@路飞的头号粉丝巴托洛米奥收到消息： 女帝汉库克想要嫁给路飞， 给路飞生猴子， 哈哈哈...
```

最后总结一下观察者模式的应用场景：当一个对象的状态发生变化，并且需要改变其它对象的时候；或者当应用中一些对象必须观察其它对象的时候可以使用观察者模式。

3. 结构图

最后画一下观察者模式对应的 UML 类图（学会观察者模式之后，要先画 UML 类图再写程序。）


Morgans
+ notify(msg : string) : void
Gossip
+ notify(msg : string) : void
NewAgency
m_list : list<Observer*>
+ attach(ob : Observer*) : void
+ detach(ob : Observer*) : void
+ notify(msg : string) : void = 0
+ ~NewAgency()
Observer
m_name : string
m_news : NewsAgency* = nullptr
+ Observer(name : string, news : NewsAgency*)
+ unsubscribe() : void
+ update(msg : string) : void = 0
+ ~Observer()



当然订阅者和发布者可能是没有子类的，因此也就不需要继承了，这个根据实际情况，具体问题具体分析就可以了。

设计模式

打赏

上一篇
策略模式 - 蒙哥·路飞

下一篇
备忘录模式 - 历史正文

相关推荐

评论

昵称 邮箱 网址(http://)

来都来了, 说点什么吧...

提交

2 评论

Anonymous Edge 111.0.1661.54 Windows 11

2023-03-27

回复

大丙愿意嫁给路飞吗

Anonymous Chrome 112.0.0.0 Windows 11

2023-05-22

回复

hello

Anonymous Chrome 108.0.0.0 Windows 11

2023-01-30

回复

爱你 大丙老师

Powered By Valine

v1.5.1