外观模式——统一编程接口

1.外观模式介绍

外观模式（Facede）在开发过程中的运用频率非常高，尤其是在现阶段各种第三方SDK充斥在我们的周边，而这些SDK很大概率会使用外观模式。通过一个外观类使得整个系统的接口只有一个统一的高层接口，这样能够降低用户的使用成本，也对用户屏蔽了很多实现细节。当然，在我们的开发过程中外观模式也是我们封装API的常用手段，例如网络模块、ImageLoader模块等。

2.外观模式定义

要求一个子系统的外部与其内部的通信必须通过一个统一的对象进行。门面模式（Facade模式）提供一个高层次的接口，使得子系统更与使用。

3.外观模式的使用场景

1. 为一个复杂子系统提供一个简单接口。子系统往往因为不断演化而变得越来越来复杂，甚至可能被替换。大多数模式使用时都会产生更多、更小的类，这使子系统更具可重用性的同时也更容易对子系统进行定制、修改，这种易变性使得隐藏子系统的具体实现变得尤为重要。Facade可以提供一个简单统一的接口，对外隐藏子系统的具体实现、隔离变化。
2. 当你需要构建一个层次结构的子系统时，使用Facade模式定义子系统中每层的入口点。如果子系统之间是相互依赖的，你可以让它们仅通过Facade接口进行通信，从而简化了它们之间的依赖关系。

4.外观模式的UML类图

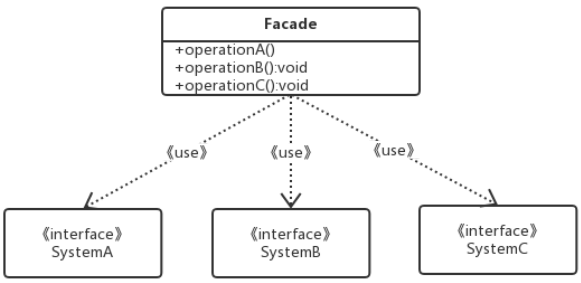

图 1

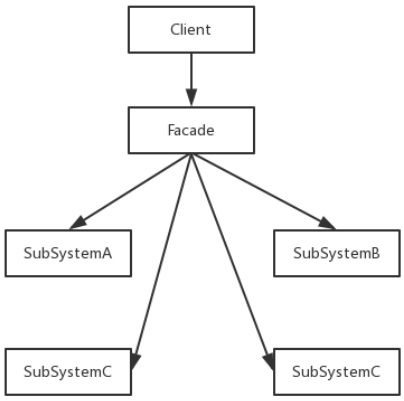

图 2

角色介绍：

- Facade：系统对外的统一接口，系统内部系统地工作。
- SystemA、SyetemB、SystemC：子系统接口（实现部分未在图1中给出）。
  外观模式接口比较简单，就是通过一个统一的接口对外提供服务，使得外部程序只通过一个类就可以实现系统内部的多种功能，而这些实现功能的内部子系统之间可能也有交互，或者说完成一个功能需要几个子系统之间进行协作，如果没有封装，那么用户就需要操作几个子系统的交互逻辑，容易出错。而通过外观类来对外屏蔽这些复杂的交互，降低用户的使用成本。它的结构如图2。

5.外观模式的简单实现

生活中使用外观模式的例子非常多，任何一个类似中央调度结构的组织都类似外观模式。举个简单的例子，手机就是一个外观模式的例子，它集合了电话功能、短信功能、GPS、拍照等于一身，通过手机你就可以完成各种功能，而不是当你打电话时使用一个诺基亚1100，要拍照时非得用一个相机，如果是这样没使用一个功能你就必须操作特定的设备，会使得整个过程很繁琐。二手机给了你一个统一的入口，集电话、上网、拍照等功能于一身，使用方便，操作简单。

下面我们来简单模拟一下手机的外观模式实现，首先我们建立一个MobiePhone类。代码大致如下.

```
1  public class MobilePhone {
2          private Phone mPhone = new PhoneImpl();
3          private Camera mCamera = new SNCamera();
4
5          public void dail() {
6                  mPhone.dail();
7          }
8
9          public void videoChat() {
10                 System.out.println("视频聊天接通中");
11                 mCamera.open();
12                 mPhone.dail();
13         }
14
15         public void hangup() {
16
```

```
                mPhone.hangup();
        }

        public void takePicture() {
                mCamera.takePicture();
        }

        public void closeCamera() {
                mCamera.close();
        }
}
```

MobilePhone类中含有两个子系统，也就是拨号系统和拍照系统，MobilePhone将这两个系统封装起来，为用户提供一个统一的操作接口，也就是说用户只需要通过MobilePhone这个类就可以操作打电话和拍照两个功能。用户不需要知道有phone这个接口以及他的实现类是PhoneImpl,同样也不需要知道Camera相关信息，通过MobilePhone就可以包揽一切。而在MobilePhone中也封装了两个子系统的交互，例如视频电话时需要先打开摄像头，然后开始拨号，如果没有这一步的封装，每次用户实现视频通过功能时都需要手动打开摄像头、进行拨号，这样会增加用户的使用成本，外观模式使得这些操作更加简单、易用。
我们看看Phone接口和PhoneImpl.

```
public interface Phone {
        // 打电话
        public void dail();

        // 挂断
        public void hangup();
}
public class PhoneImpl implements Phone{

        @Override
        public void dail() {
                System.out.println("打电话");
        }

        @Override
        public void hangup() {
                System.out.println("挂电话");
        }

}

```

代码很简单，就是单纯的抽象与实现。Camera也是类似的实现，代码如下：

```

public interface Camera {
        public void open();

        public void takePicture();

        public void close();
}
```

```
9
10  public class SNCamera implements Camera {
11
12          @Override
13          public void open() {
14                  System.out.println("打开相机");
15          }
16
17          @Override
18          public void takePicture() {
19                  System.out.println("拍照");
20          }
21
22          @Override
23          public void close() {
24                  System.out.println("关闭相机");
25          }
26
27  }
```

测试代码：

```
1   public class Test {
2           public static void main(String[] args) {
3                   MobilePhone phone1 = new MobilePhone();
4                   // 拍照
5                   phone1.takePicture();
6                   // 视频聊天
7                   phone1.videoChat();
8           }
9   }
10
```

运行结果：

```
1   拍照
2   视频聊天接通中
3   打开相机
4   打电话
```

从上述代码中可以看到，外观模式就是统一接口封装。将子系统的逻辑、交互隐藏起来，为用户提供一个高层次的接口，使得系统更加易用，同时也对外隐藏具体的实现，这样即使具体的子系统发生改变，用户也不会感知到，因为用户使用的是Facade高层接口，内部的变化对于用户来说并不可见。这样一来将变化隔离开来，使得系统也更为灵活。

总结

外观模式是一个高频率使用的设计模式，它的精髓就在于封装二字。通过一个高层次结构为用户提供统一的API入口，使得用户通过一个类型就基本能够操作整个系统，这样减少了用户的使用成本，也能够提升系统的灵活性。

优点

1. 对客户程序隐藏子系统的细节，因而减少了客户对于子系统的耦合，能够拥抱变化。
2. 外观类对子系统的接口封装，使得系统更易于使用。

缺点

1. 外观类接口膨胀。由于子系统的接口都由外观类统一对外暴露，使得外观类的API接口较多，在一定程度上增加了用户的使用成本。
2. 外观类没有遵循开闭原则，当业务出现变更时，可能需要直接修改外观类。

xyzso1z
原创文章 80　获赞 40　访问量 2万+