**观察者模式的详解：**

1. 我们有三个窗体，Form1、Form2、Form3，这时候我们有一个需求：当Form1窗体中的文本控件发生变化的时候，From2、From3中的文本控件也会随之发生变化，这时候我们就需要用到观察者模式的设计理念去设计这个程序。我们可以把Form1当作是被观察者（Subject），然后让Form2、Form3当成是一观察者，通过委托（不推荐）、事件、接口…方式让观察者去注册被观察者的信息源，当被观察者Form1发生相应的改变的时候，通知观察者Form2、From3，然后观察者From2、Form3也发生着相应的改变
2. 简而言之，就像是生活中的微博，某个博主就是一个被观察者（Subject），而关注他的人则是观察者。当被观察者发生相应的改变的时候，如：发微博…，观察者也能够接收到他的信息源，从而自己也发生一些改变

**中介者模式：**

1. 我们有两个窗体Form1，Form2，现在Form1需要对Form2进行一些都写的操作，当我们删除掉了Form1中的一些代码、Form2中的一些代码或者是我们现在又增加了一个窗体Form3让Form1与之进行操作。这时候我们会发现他们三者之间的耦合度太高了，因为我随便删除一个都可能会出现错误，或者是我又增加一个窗体的时候Form1需写很多代码，这时候就出现了一个中介者的模式。我们通过中介者模式的设计理念，去建立一个中介者的类，我们让Form1对Form2、Form3的读写操作都在中介者的类中执行，然后中介者类又把相应的操作传导给了Form2、Form3，我们通过一个中介者的身份，可以完美的隔开了Form1与Form2、Form3之间的耦合度，以至于在Form1中不出现关于Form2、Form3的代码也能对这两个窗体进行操作
2. 就好像是房产中介，当我们去租房子的时候，我们面对的是中介而不是房东，当我们给房东房租亦或是办理关于房东的相关手续的时候也是通过中介来完成的，这样中介就起到了一个作用，就算我不见面房东，我也能够把房子租下来

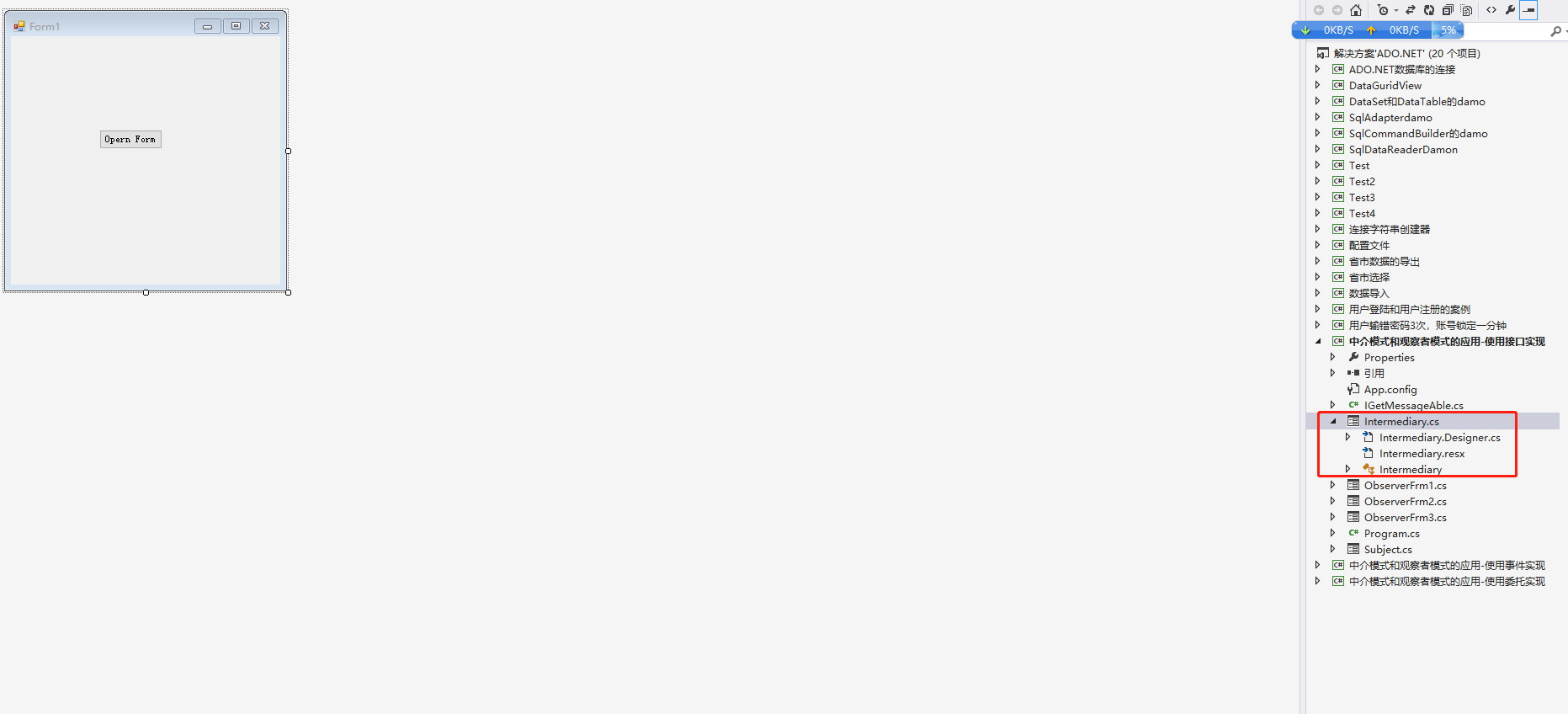
**观察者模式与中介者模式的联合使用实现窗体之间的传值**

**问题解析：**

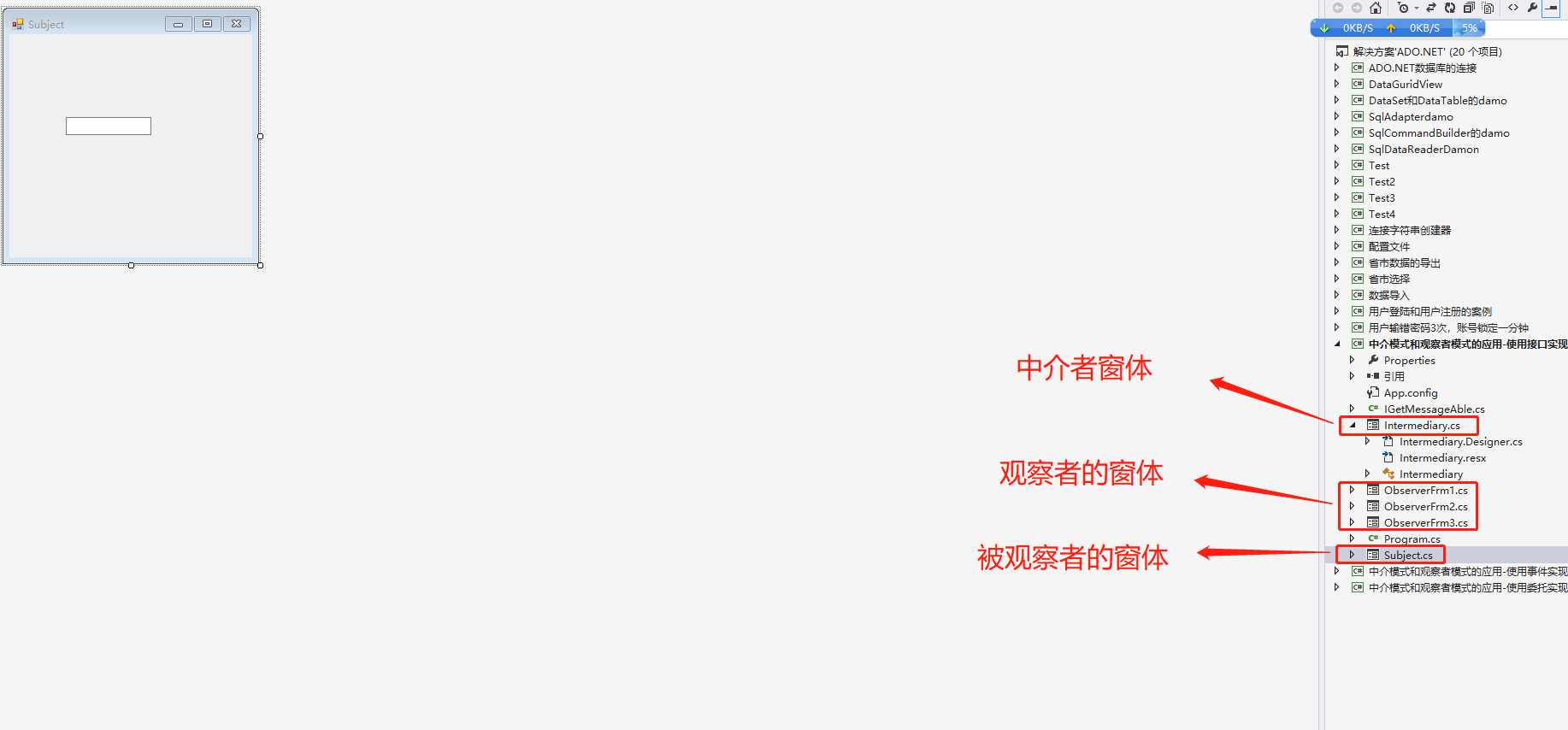
1. 该思想能够使用委托、事件、接口…来实现，这里只演示接口的用法，其他使用方法可以在 源码 -> ADO.NET中查看
2. 中介者模式和观察者模式的配合使用可以实现观察者和被观察者的彻底解耦

**具体实现：**

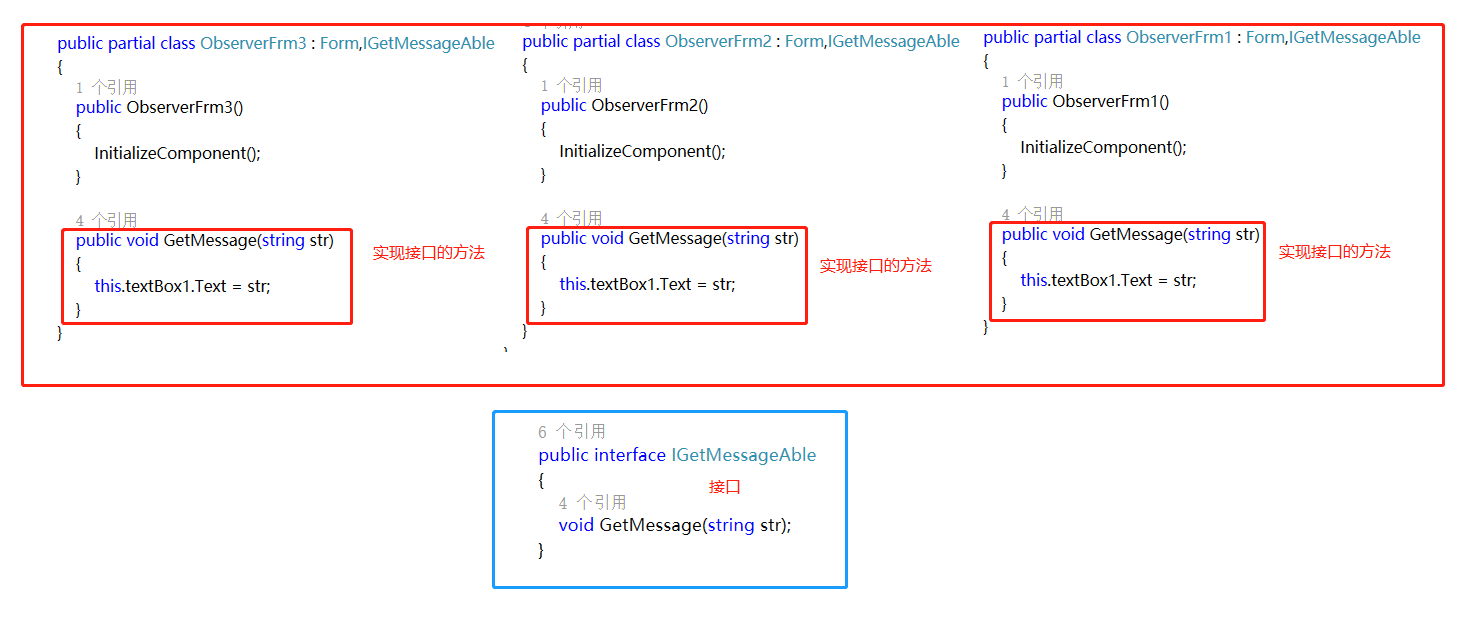
1. 我们先建立一个中介者的窗体，该窗体有一功能是负责打开观察者窗体和被观察者窗体，主要功能是实现观察者窗体与被观察者窗体的交互能通过中介者来实现，使被观察者窗体中不出现观察者的窗体的代码也能够实现两者间的交互



1. 我们建立好一被观察者窗体，和诸多观察者窗体，这些窗体都拥有一个文本框的控件。我们现在的需求是，当被观察者窗体在文本框控件中输入文本的时候，观察者窗体中的文本框控件也显示出相应的文本



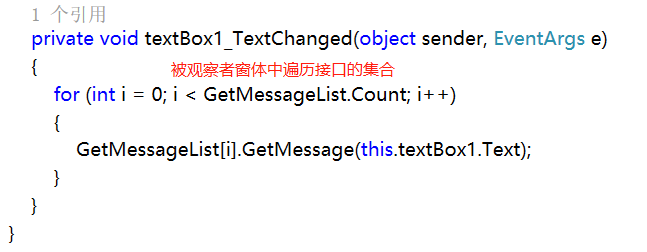
1. 我们定义一个接口，并在接口中写一个相应的方法，该方法没有返回值，有一个参数，然后让观察者继承并实现这个接口中的函数



1. 当观察者们都继承并实现了这个接口后，子类可以直接赋值给父类，我们就可以让被观察窗体中定义一个接口的集合，然后通过在中介者窗体加载完成的时候，创建被观察者窗体和观察者窗体的对象，并且往被观察者所定义的接口集合中添加继承接口的观察者的对象



1. 这时候，因为观察者的对象都放在了被观察者窗体中定义的接口集合中，并且都实现了接口中的函数，所以，当我们调用接口的对象的时候，其实是在调用赋值给接口的观察者的对象，也就是说，我们可以通过接口对象来调用观察者实现接口的函数，所以，我们可以在被观察者文本框控件内容发生改变事件里遍历这个接口的集合，并且逐一的调用观察者实现接口的函数



1. 通过这样，我们其实是在调用的是观察者所执行的函数，但是我们却传了一个被观察者文本框的文本过去给观察者实现接口的函数做参数，这样我们就通过了中介者模式和观察者模式的思想实现了窗体之间的传值
2. 我们会发现，在被观察者窗体中的代码中并没有出现与观察者相关的代码，这就是中介者的好处，中介帮我们实现了观察者与被观察者在不知道对方是谁的情况下的交互，彻彻底底的解开了观察者与被观察者之间的耦合度！