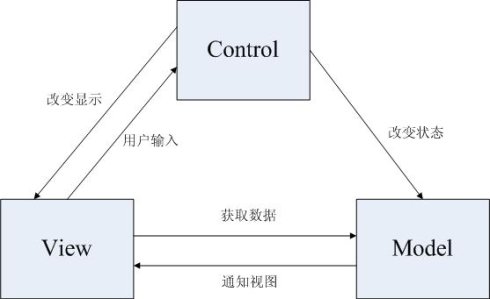
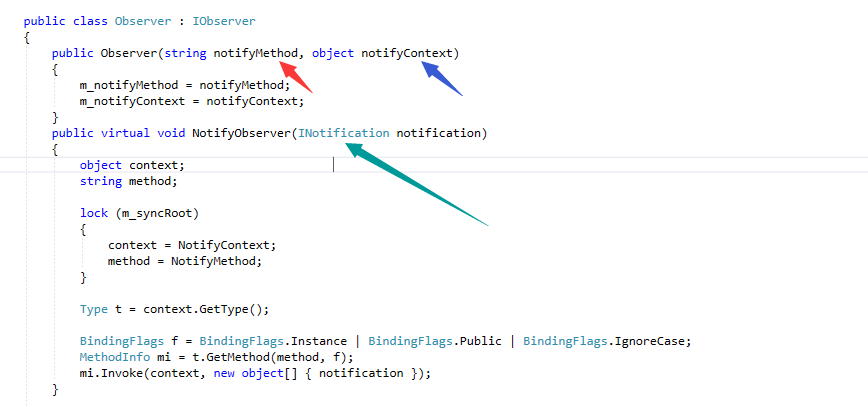
**[快速上手并深入理解PureMVC框架设计(C#)](https://www.cnblogs.com/chenggg/p/10743739.html)**

直接进入正题谈谈框架之间是如何解耦吧...



首先我们先分析下辅助工具类Observer类



可以看到在构造观察者时 传入了对象和对象包含的方法(蓝色箭头为对象,红色箭头为对象内包含的方法)

接下来我们查看观察者的执行(NotifyObserver)

参数是框架给我们提供的标准(Inotification接口 通知名称(name),通知的执行对象(body))

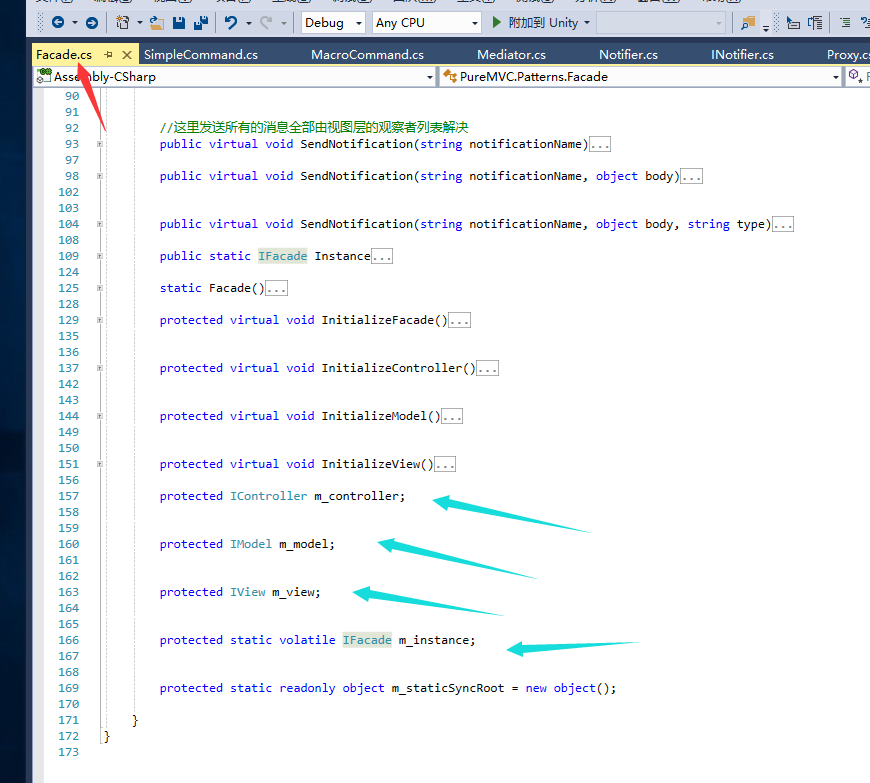
在执行观察者的时候 反射出执行对象的方法 然后调用方法

同时PureMVC中已经为我们统一化了发送标准和消息体标准(Notifier,Notification)

发送者(Notifier)中包含了外观模式的单例引用(Facade)  然后中介者(Mediator),代理者(Proxy) 和 命令(Command)中统一继承发送者即可全部实现发送消息的标准

↑词条信息很重要

再来看看Facade



可以看到Facade持有对其他三个层中单例的引用  同时Facade本身以单例的形式存在

接下来 我们进入Facade主要功能看看有哪些



可以看到此时Facade囊括了MVC三层能做到的所有功能

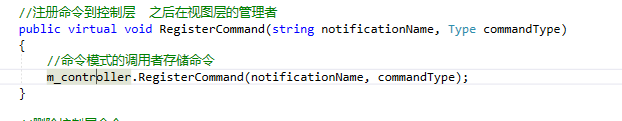
接下来 我们尝试解读在这种外观模式的封装后如何使用(边看边分析)



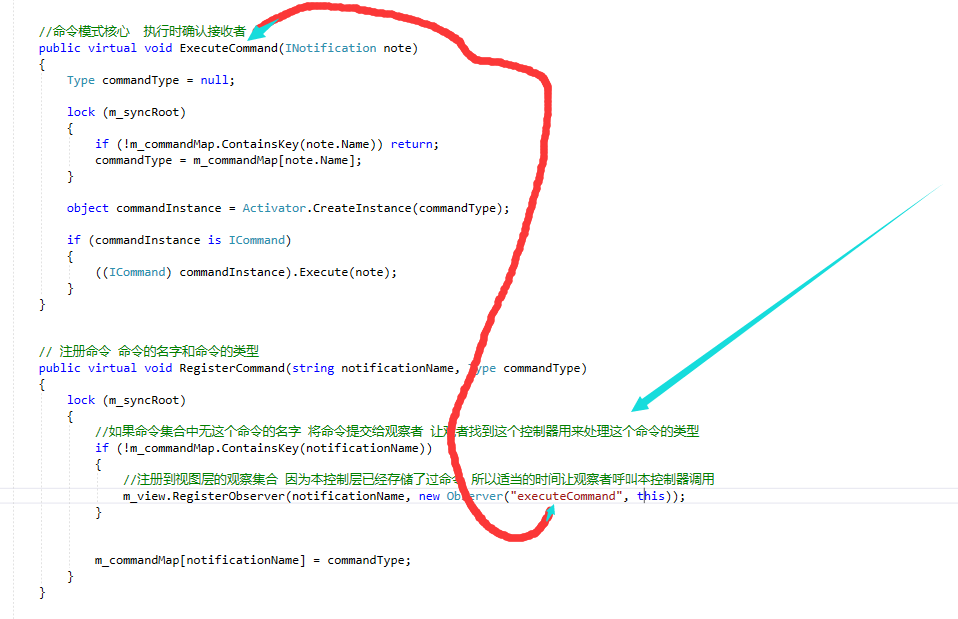
我们在一个Facade的子类中进行了 命令的注册 中介者的注册 以及代理者的注册

在我们完成每一次注册之后 PureMVC框架为我们做了哪些操作?

首先是注册命令 我们可以看到Facade在使用控制器进行命令注册



然后



注意 上文中讲到的Observer 此时发挥作用 可以看到在注册命令的时候 我们将此条命令推送给了视图层的观察者集合



 一个事件(notificationName)此时对应了一个观察列表 也就是说这种设计再次对方法进行了解耦和  \*\*\*\*\*\*\*重点\*\*\*\*\*\*\*

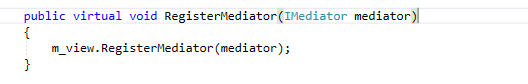
 通过控制层把消息推送给了视图层  而视图层又将消息添加到了观察者集合中

 此时我们的注册命令阶段结束

 接下来看看注册中介时 框架为我们做了哪些操作



首先进入Facade注册中介者



 进入视图层 接下来分析四个箭头的意义



首先是蓝色箭头 可以看到在注册中介者时调用了ListNotificationInterests方法拿到一个字符串集合

这里也就是说 你的XX中介者类中一定会存在这么一个名为IList<string> ListNotificationInterests();的方法 \*\*\*\*\*\*\*\*\*\*重点\*\*\*\*\*\*\*\*\*\*\*

接着框架为我们构造了一个观察者对象 并且我们注册的中介者现在是一个方法的执行者 执行的方法名称" handleNotification"(红色箭头指向的)

也就是说框架规定了我们在写XXX中介者类时必须包含一个名为"handleNotification"的方法

同时这个观察对象被关联到了很多个观察事件中

上文已经做出分析 观察者通过对象的反射执行对象的方法

那么现在这个观察者对这堆事件都做出了不同的处理

这就达到了中介者设计模式的初衷! 让网状结构转化为星状结构

这样的话我们一个方法就要处理很多不用的事件 也就是把网状结构的耦合度向星状图的原点输送

方法中处理起来的话  必然是判断了 (switch 或者if)

这样的话我们必须先表明事件添加到List中

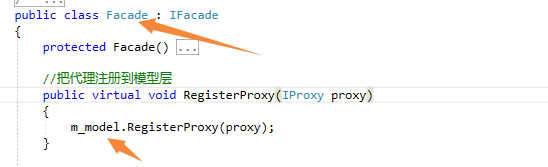
此时完成注册 我们注册时 框架提供观察者记录了我们中介者事件中所有的事件

ps:紫色箭头这种东西有很多  比如我们 注册中介 命令 模型 的Register和Remove 还有modle层 view层 controll的Initialize方法

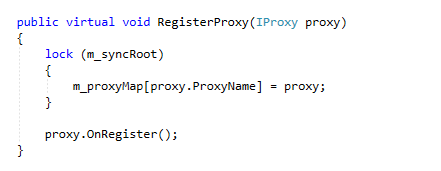
　　 这个主要作用是做一些额外的操作  比如说 有的视图层在做视图操作或注册命令时会有UI表现有的又没有

　　 此时通过这种方法解耦 让你很舒服的处理这种情况

接下来我们可以看到代理者的注册



进入模型层 可以代理注册很简单 就是把数据塞入字典



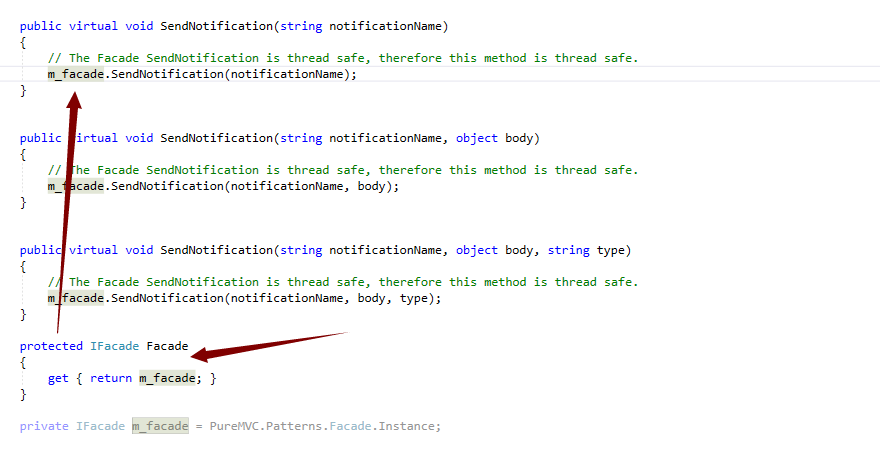
 那么做完了三个层的注册 操作  接下来分析 在触发事件之后框架又为我们做了哪些操作

 举个例子 在注册中介者的时候 我们在中介者的构造阶段给一个按钮绑定一个事件 然后尝试去触发他

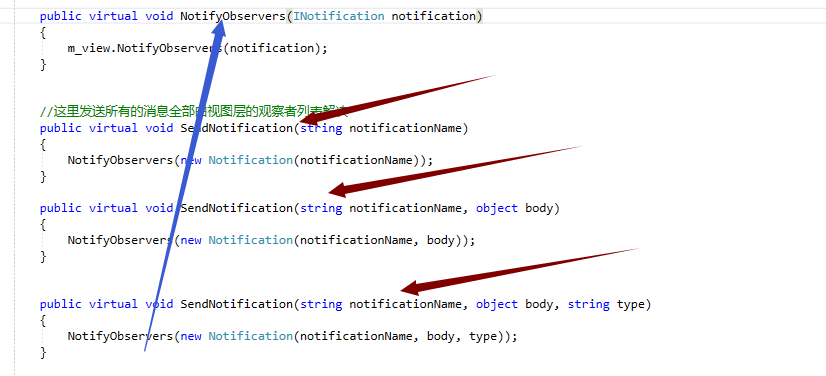


在上面已经总结了 我们的三大类 中介 命令 代理 都被框架统一化了发送通知的标准(继承Notifier 使用其中SendNotification方法)

所以每个功能 触发命令的时候 直接调用SendNotification方法  接下来 我们查看触发方法之后 框架又做了哪些操作



进入外观模式单例



此时进入了视图层的方法处理模块 我们进入这个模块对其功能进行分析



可以看到方法中先获取一个事件对应的几个观察者 然后对其执行(上文中提到的 观察者持有观察对象和对象内的方法  通过反射进行方法执行)

Ok 此时框架全自动执行了 观察中心的方法

接下来总结一下此间几种模式的作用

外观模式:就如项目中所见 对几个层进行封装目的在于让使用者觉得边界  这一点SendNotification方法效果最为显著

命令模式:对方法进行解耦 让方法变得无限单一

观察者:是一所有事件的处理中枢

中介者:将一个系统的耦合度转化到一个点上

代理模式:主要是统一对数据的处理标准  对数据和数据的处理方法解耦和

单例：...