wpf模式窗口:WPF中MVVM模式原理分析和实战

疯狂代码 http://CrazyCoder.cn/ ĵ:http:/CrazyCoder.cn/DotNet/Article66346.html

1, 前提

可以说MVVM是专为WPF打造模式, 也可以说MVVM仅仅是MVC个变种, 但无论如何, 就实战而言, 如果你或你团队(Team)没有使用"Binding"习惯, 那么研究MVVM就没有多大意义.

另外,个人觉得, 使用Command以及打造种合理简化方式去使用Command也和使用Binding样重要.

2, 诞生

为了解决现实世界中问题,我们需要将现实世界中事物加以抽象, 然后得到了Do Object, 无论贫血还是富血, 我们都可以简单地把他们归结为"由现实世界抽象出来模型", 也就是我们model, 也就M-V-VM中"M"

但其无法和我们用户进行交互, 所以, 我们需要为其创建个界面(视图, View), 该视图可以和用户输入设备进行交互, 这很棒, 但问题是如何将View和我们model关联起来? Binding便可以发挥作用了, 比如视图上某个文本框中文本和Model中"用户名"关联起来, 用户便可以通过操作该文本框来访问和修改Model"用户名"了

这是极其简单情况, 但实际编程时我们发现, Model中属性(和思路方法)往往不那么容易和View中界面Control控件关联起来, 比如, "类型不匹配": 界面Control控件所需要类型和模型中属性提高类型不匹配. "需要额外操作": 模型中数据需要经过些额外处理才能传给视图,反的亦然. 此时, 我们意识到View似乎需要个"Helper"类来处理些额外工作.

这个helper所包含代码可以放在除了Model外很多地方(我们现在不考虑贫血富血的类争论), 比如View中, 记得自己刚学习窗体开发时就是这么干, 将绝大多数处理逻辑放在那个所谓CodeBehind中. 后来,正如大家在各种设计模式书籍中所看到样,为了将View和Model剥离开来,实现view可替换(比如你可以讲自己精心设计软件Software同时运行于窗体,Web甚至Mobile上), 便有了MVC. 有了MVC以后似乎就开始滋生M-V-XXX的类争论和变种模型, 比如MVP以及这里MVVM,甚至MVP也有着Supervising Controller和Presentation Model两种方式. 但主要围绕两个问题,是model和view的间关系, 完全隔离?单向还是双向? 2是这个"XXX"需要完成哪些功能,简单流程调度?复杂规则处理? OK,这些争论都没有关系, 是否采用某种模式取决于你开发所处环境(比如语言特性,框架特性)以及你业务特性以及所面临主要变化点等等

但和MVC,MVP所区别是,MVVM引入不仅仅是技术上原因(解除耦合应对变化等老生常谈),另外个很大原因是:软件Software团队(Team)开发方式改变.如果你做过段时间WPF项目开发话,你可能会有比较明显感觉:在View层打造上,如何分配员和美工工作.在继续阅读的前,大家可以看看我以前篇文章"在UI Designer和

Developer的间". 以前我们团队(Team)采用便是"集成模式", 我便兼职了其中"Integrator"角色.这还不错.但说实在,这仅仅是个在特殊情况下不得已而为的暂时方案,所以我们付出了很大努力开始转向"收割模式"了,要转向这个模式,至少需要两个基本条件:

(1)你拥有能够熟练运用Blend等工具能为员输出XAML美工, 他专注于纯粹UI/UE, 另外他还必须具有定"员"思维.以便输出东西能很好地作为部分而运转起来,而不是仅仅"看上去"是那样

(2)你需要能够脱离View层但仍能编写出高质量代码员

幸运是, 我们在努力创造条件1,并取得了很好效果.(你可以招个具有Flash脚本编写经验并且有极大学习热情美工人员, 并对他进行Blend相关培训). 而MVVM模式为我们实现第 2个条件提供了极大便利. 为什么MVC/MVP模式不行而MVVM可以呢? 很简单, 在MVC和MVP模式中, View层都具有很多代码逻辑, 开发View层是员, 虽然UI/UE团队(Team)会做很多工作, 但这个层"实现者"仍然是员. 在以前开发中,其工作得很好, 而在WPF开发中员对View层展现显得力不从心了,美工(指符合上面条件1美工)虽然很擅长, 但他会说"可惜我不会".于是, 我们需要种方式将View层代码逻辑抽取出来,并View层很纯粹以便完全让美工去打造它.相应地, 需要将View层相应逻辑抽取到个代码层上,以便让员专注在这里

回想下, 我们只所以要在View(Xaml)背后写些代码(C#), 无非是想传递些数据以及传递数据时数据处理或在用户和界面Control控件进行交互时执行些操作, 最简单例子是在MVC中当界面发生交互时View去Controler中某个思路方法, 以便将该操作相应"指示"传递到"后台"去. 在以前技术中, 这样"衔接性"代码是必须. 而在WPF中, 则可以通过另外技术来进行层和层的间"衔接", 这就是"Binding" 和"Command", 以及稍后我们会提到"AttachBehavior". 通过Binding, 我们可以实现数据传递; 通过Command, 我们可以实现操作.(AttachBehavior作用稍后再谈). Binding和Command是可以写在XAML中, 这样看来XAML后面对于CS文件可以被完全抛弃或不予理会了. 这样XAML文件正是美工所需要. 而这些对于Binding以及Command定义描述以及其他相关信息代码应该放在那里呢, 当然不是View, 更不是Model, 是"ViewModel". ViewModel是为这个View所量身定制, 它包含了Binding是所需相关信息,比如Converter以及为ViewBinding提供DataContext, 它包含了Command定义以便View层可以直接使用, 另外,它还是个变种Controler, 它得负责业务流程调度

于是, 便有了这副图, 然后, 正如"时势造英雄"所言, MVVM就诞生了.

3, ViewModel 和单元测试

如果你是名正在使用MVVM模式打造软件Software员, 那么我劝你尽快忘掉View. 你所面对是这样个模式"UnitTest-ViewModel-Model"(这并非个模式, 仅仅是我为阐述观点而暂时如此表述)

记得曾经有个Model-View-AbstractView模式, 而MVVM中VM实际也是个AbstractView: the

abstraction of view. 它是个抽象View, 具有个View灵魂,而不具备相应可视化Control控件而已. 所以对于员而已, 打造这样个抽象VM就可以认为是完成View层打造了.而当美工完成无数Control控件组成实际View后, 我们就可以用Binding和Command这样黏合剂将这个抽象View和实际View黏合在起了

那么在黏合的前, 我们如何知道自己VM是否正常工作呢? 单元测试!

在介绍说明对于ViewModel进行单元测试重要性的前, 送给大家句话: "View and Unit Test are just two dferent types of ViewModel consumers" (Josh Smith). 如果我们将ViewModel看作生产者, 那么View和Unit Test都是具有同等地位消费者而已. 并且UnitTest相比于View而言具备更大消费能力. 或者你可以简单认为View也仅仅是种不太推荐测试方式而已. 所以要实施好这个模式, 那么对ViewModel单元测试就是必须了,并且这个测试要不依赖于任何UIControl控件. (那么不是不对应ViewModel开发是不是就应该通过测试来驱动了?TDD?)

4, AttachBehavior

般情况下利用Command, Binding, AttachProperty等WPF特性, View和ViewModel的间能配合工作得很好. 假设我们有个Button, 当该Button被点击时候我们要完成些操作, 很简单, 将该操作封装成个Command并绑定到该Button上就可以了, 但如果我们要在Button被Load时候执行另外些操作呢? 由于Button没有直接被Load事件所触发Command, 所以不能使用Command了. 不能直接将Load事件处理器写在Button所在Xaml所对应CS文件里, 这和我们刚才对MVVM设计是相矛盾. 个不太好方案是继承下Button, 并撰写个由Load所触发Command, 这可行, 但明显不好. 正如个Control控件没有某个属性并且在不继承情况下而采用AttachProperty样, 我们可以采用AttachBehavior. AttachBehavior不是WPF特性, 它仅仅是个最佳实战, 个Pattern. 有关AttachBehavior语法如何书写, 请参考 : http://www.codeproject.com/KB/WPF/AttachedBehaviors.aspx

2009-9-2 0:01:40
疯狂代码 http://CrazyCoder.cn/