# 22 | 桌面程序的架构建议

2019-07-05 许式伟

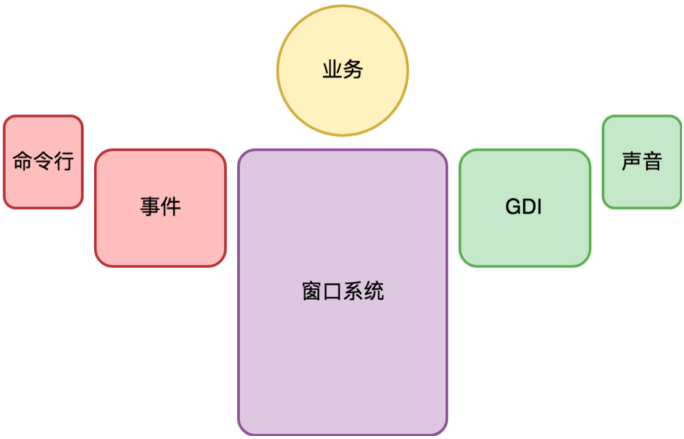
许式伟的架构课 进入课程 >

讲述：姚迪迈

时长 12:42 大小 11.65M

你好，我是七牛云许式伟。

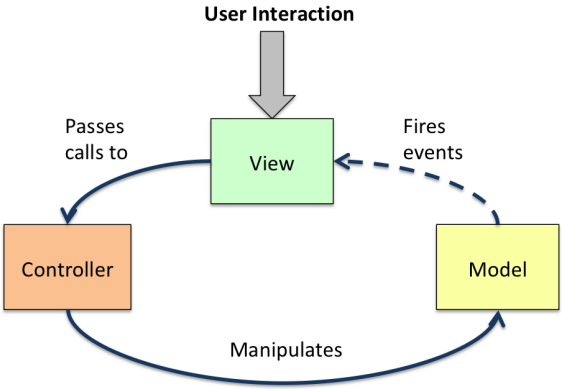
上一讲我们介绍了图形界面程序的框架。站在操作系统交互子系统的角度来看，我们桌面应用程序的结构是下面这样的。




今天我们换一个角度，站在应用架构的角度，来聊聊如何设计一个桌面应用程序。

## 从 MVC 说起

关于桌面程序，我想你听得最多的莫过于 MVC 这个架构范式。MVC 全称是 “模型 (Model)- 视图 (View)- 控制器 (Controller)” 。




**Model-View-Controller**

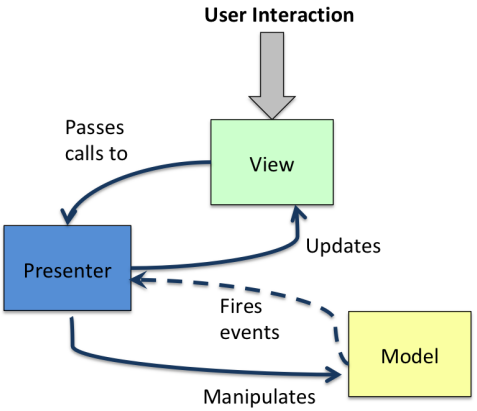
怎么理解 MVC 呢？一种理解是，Model 是 Input，View 是 Output，Controller 是 Process，认为 MVC 与计算机的 Input-Process-Ouput 这个基础模型暗合。

但更准确的解释是：Model 是数据，View 是数据的显示结果，同时也接受用户的交互动作，也就是事件。从这个意义来说，说 Model 是 Input 并不严谨，View 接受的用户交互，也是 Input 的一部分。

Controller 负责 Process（处理），它接受 “Model + 由 View 转发的事件” 作为 Input，处理的结果（Output）仍然是 Model，它更新了 Model 的数据。

View 之所以被理解为 Output，是因为 Model 的数据更新后，会发送 DataChanged（数据更新）事件，View 会在监听并收到 DataChanged 事件后，更新 View。所以把 View 理解为 Output 也并不算错，它从数据角度看其实是 Model 的镜像。

对 MVC 模式做些细微的调整，就会产生一些变种。比如，Model 的数据更新发出 DataChanged 事件后，由 Controller 负责监听并 Update View，这样就变成了 MVP 架构。MVP 全称是 “模型 (Model)- 视图 (View)- 表现 (Presenter)” 。




Model-View-Presenter

那么，我们究竟应该选择哪一种架构范式比较好？

要想判断我们写的程序架构是否优良，那么我们心中就要有架构优劣的评判标准。比较知名且重要的一些基本原则如下。

> 最低耦合原则：不同子系统（或模块）之间有最少的交互频率，最简洁且自然的接口。
>
> 单一职责原则：不要让一个子系统（或模块）干多件事情，也不要让它不干事情。

如果在我们心中以遵循架构法则为导向，回过头再来看 MVC，又会有不同的理解。

## 理解 Model 层

我们先看 Model。如果你真正理解 Model 层的价值，那么可以认为你的架构水平已经达到了较高层次的水准。因为 Model 层太重要了。

我上面说 Model 层是数据，这其实还不是太准确。更准确来说，Model 层是承载业务逻辑的 DOM，即 “文档对象模型（Document Object Model）” 。直白理解，DOM 是 “面向对象” 意义上的数据。它不只是有数据结构，也有访问接口。

为了便于理解，假设我们基于数据库来实现 Model 层。**这种情况下会有两种常见的架构误区。**

一种是直接让 Controller 层直接操作数据库，也就是拿数据库的读写接口作为 Model 层的接口。

另一种看起来高级一些，用所谓的 ORM 技术来实现 Model 层，让 Controller 直接操作 ORM。

为什么我们说这两种做法都有问题呢？原因就在于对 Model 层的价值不明。Model 层的使用接口最重要的是要自然体现业务的需求。

只有这样，Model 层的边界才是稳定的，与你基于的技术无关。是用了 MySQL，还是用了 NoSQL？是直接裸写 SQL 语句，还是基于 ORM？这都没关系，未来喜欢了还可以改。

另外，从界面编程角度看，Model 层越厚越好。为什么这么说？因为这是和操作系统的界面程序框架最为无关的部分，是最容易测试的部分，也同时是跨平台最容易的部分。

我们把逻辑更多向 Model 层倾斜，那么 Controller 层就简洁很多，这对跨平台开发将极其有利。

这样来看，直接让 Controller 层直接操作数据库，或者基于 ORM 操作数据库，都是让 Model 层啥事不干，这非常非常浪费，同样也违背了 “单一职责原则”。

我们需要强调，单一职责不只是要求不要让一个子系统（或模块）干多件事情，同时也要求不要让它不干事情。

如果我们用一句话来描述 Model 层的职责，那么应该是“负责业务需求的内核逻辑”，我们以前经常叫它“DataCore”。

那么 Model 层为何要发出 DataChanged 事件?

这是从 Model 层的独立性考虑。Model 层作为架构的最底层，它不需要知道其他层的存在，不需要知道到底是 MVC 还是 MVP，或者是其他的架构范式。

有了 DataChanged 事件，上层就能够感知到 Model 层的变化，从而作出自己的反应。

如果还记得第一章我们反复强调的稳定点与变化点，那么显然，DataChanged 事件就是 Model 层面对需求变化点的对策。大部分 Model 层的接口会自然体现业务需求，这是核心价值点，是稳定的。

但是业务的用户交互可能会变化多端，与 PC 还是手机，与屏幕尺寸，甚至可能与地区人文都有关系，是多变的。

用事件回调来解决需求的变化点，这一点 CPU 干过，操作系统也干过，今天你做业务架构也这么干，这就很赞。

## 理解 View 层

View 层首要的责任，是负责界面呈现。界面呈现只有两个选择，要么自己直接调用 GDI 接口自己画，要么创建子 View 让别人画。

View 层另一个责任是被自然带来的，那就是：它是响应用户交互事件的入口，这是操作系统的界面编程框架决定的。比较理想的情况下，View 应该把自己所有的事件都委托（delegate）出去，不要自己干。

但在 View 的设计细节中，也有很多问题需要考虑。

**其一，View 层不一定会负责生成所有用户看到的 View。**有的 View 是 Controller 在做某个逻辑的过程中临时生成的，那么这样的 View 就应该是 Controller 的一部分，而不应该是 MVC 里面的 View 层的一部分。

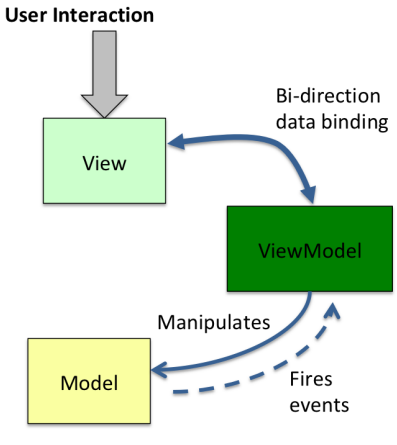
**其二，View 层可能需要非常友好的委托（delegate）机制的支持。**例如，支持一组界面元素的交互事件共同做委托（delegate）。

**其三，负责界面呈现，意味着 View 层和 Model 层的关系非常紧密，紧密到需要知道数据结构的细节，这可能会导致 Model 层要为 View 层提供一些专享的只读访问接口。**这合乎情理，只是要确保这些访问接口不要扩散使用。

**其四，负责界面呈现，看似只是根据数据绘制界面，似乎很简单，但实则不简单。**原因在于：为了效率，我们往往需要做局部更新的优化。如果我们收到 onPaint 消息，永远是不管三七二十一，直接重新绘制，那么事情就很好办。但是在大部分情况下，只要业务稍微复杂一点，这样的做法都会遇到性能挑战。

在局部更新这个优化足够复杂时，我们往往不得不在 Model 和 View 之间，再额外引入一层 ViewModel 层来做这个事情。

ViewModel 层顾名思义，是为 View 的界面呈现而设计的 Model 层，它的数据组织更接近于 View 的表达，和 View 自身的数据呈一一对应关系（Bidi-data-binding）。



Model-View-ViewModel

一个极端但又很典型的例子是 Word。它是数据流式的文档，但是界面显示人们用得最多的却是页面视图，内容是分页显示的。

这种情况下就需要有一个 ViewModel 层是按分页显示的结构来组织数据。其中负责维持 Model 与 ViewModel 层的数据一致性的模块，我们叫排版引擎。

从理解上来讲，我个人会倾向于认为 ViewModel 是 View 层的一部分，只不过是 View 层太复杂而进行了再次拆分的结果。也就是说，我并不倾向于认为存在所谓的 “Model-View-ViewModel” 这样的模式。

## 理解 Controller 层

Controller 层是负责用户交互的。可以有很多个 Controller，分别负责不同的用户交互需求。

这和 Model 层、View 层不太一样。我们会倾向于认为 Model 层是一个整体。虽然这一个层会有很多类，但是它们共同构成了一个完整的逻辑：DOM。而 View 层也是如此，它是 DOM 的界面呈现，是 DOM 的镜像，同样是一个整体。

但负责用户交互的 Controller 层，是可以被正交分解的，而且应该作正交分解，彼此完全没有耦合关系。

一个 Controller 模块，可能包含一些属于自己的辅助 View，也会接受 View 层委托的一些事件，由事件驱动自己状态，并最终通过调用 Model 层的使用接口来完成一项业务。

Controller 模块的辅助 View 可能是持续可见的，比如菜单和工具条；也可能是一些临时性的，比如 Office 软件中旋转图形的控制点。

对于后者，如果存在 ViewModel 层的话，也有可能会被归到 ViewModel + View 来解决，因为 ViewModel 层可以有 Selection 这样的东西来表示 View 里面被选中的对象。

Controller 层最应该思考的问题是代码的内聚性。哪些代码是相关的，是应该放在一起的，需要一一理清。这也是我上面说的正交分解的含义。

如果我们做得恰当，Controller 之间应该是完全无关的。而且要干掉某一个交互特别容易，都不需要删除该 Controller 本身相关的代码，只需要把创建该 Controller 的一行代码注释掉就可以了。

从分层角度，我们会倾向于认为 **Model 层在最底层；View 层在中间**，它持有 Model 层的 DOM 指针；**Controller 层在最上方**，它知道 Model 和 View 层，它通过 DOM 接口操作 Model 层，但它并不操作 View 去改变数据，而只是监听自己感兴趣的事件。

如果 View 层提供了抽象得当的事件绑定接口，你会发现，其实 Controller 层大部分的逻辑都与操作系统提供的界面编程框架无关（除了少量辅助 View），是跨平台的。

**谁负责把 MVC 各个模块串起来呢？当然是应用程序（Application）了。**在应用开始的时候，它就把 Model 层、View 层，我们感兴趣的若干 Controller 模块都创建好，建立了彼此的关联，一切就如我们期望的那样工作起来了。

## 兼顾 API 与交互

MVC 是很好的模型来支持用户交互。但这不是桌面程序面临的全部。另一个很重要的需求是提供应用程序的二次开发接口（API，全称为 Application Programming Interface）。

提供了 API 的应用程序，意味着它身处一个应用生态之中，可以与其他应用程序完美协作。

通过哪一层提供 API 接口？我个人会倾向于认为最佳的选择是在 ViewModel 层。Model 层也很容易提供 API，但是它可能会缺少一些重要的东西，比如 Selection。

## 结语

这一讲我们探讨了一个桌面应用程序的业务架构设计。我们探讨了大家耳熟能详的 MVC 架构范式。一千个人眼中有一千个哈姆雷特，虽然都在谈 MVC，但是大家眼中的 MVC 各有不同。

我们站在什么样的架构是好架构的角度，剖析了 MVC 的每一层应该怎样去正确理解与设计，有哪些切实的问题需要去面对。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲我们将聊聊基于浏览器的开发。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。



上一篇　21 | 图形界面程序的框架

## 精选留言 (22)

写留言

**Geek_架构师**
2019-07-05

比较抽象，许老师，有没有比较好的实例程序推荐，通过程序来理解应用架构的具体细节？

作者回复: 后面会考虑讲一个例子

9

**黎**
2019-07-05

唯一每天12点等更新的专栏

展开

6

**勇闯天涯**
2019-07-05

经过许老师一番分析，对MVC的理解更深刻了，明天到公司把ViewModel重新捋捋。

3

**马哲富**
2019-07-05

许老师好！

工作中也经常用到MVC模式开发，经常用个Mode层就是一个和数据库映射的实体，然后再View层和Controller层传输数据，不知道老师文中所指的Mode层应该是“负责需求的内核逻辑”应该如何理解？难道需...

展开

作者回复: 你说的service，应该就是我说的model层。一些说法是把model层分为service和DAO层，但是实际上DAO根本算不上一层。

2

**小智e**

2019-07-06

开始不理解了，需要多经历，然后再回头来看

展开

作者回复: 是的，这本书因为内容高度浓缩，需要反复看。这是练内功与练外功的区别。练外功可能有学完的那天，内功没法学完，甚至可能你会发现比书上更好的体悟。

1

**Smallfly**

2019-07-06

这几种模式中，对 MVVM 理解是最多样化的，很大一部分原因是取决于原来是如何使用 MVC 的，可以分为三种流派：

1、在 Controller 中处理所有的业务逻辑，包括监听...

展开

作者回复: 挺好的补充

1

**1900**

2019-07-05

“文档对象模型”中的“文档”应该如何理解？是因为linux中“一切皆文件”，所以这里一切皆文档么？

我目前只能理解“对象”和“模型”，对象指数据+操作，数据对应了结构体（数据结构），操作对应了方法...

展开

作者回复: 这个是一个惯例，一般对象模型是一颗树，树根叫Document，所以叫DOM（文档对象模型）。xml的根对象就是Document是同样的道理。

1

**靠人品去赢**

2019-07-05

老师讲到了MVC和MVVM构架，我理解的前后端分离是一个趋势，model不单单是数据model，同样view也不单单只是用来展示。实际上要把control这个拆开，view也需要control，他可以是有向后台发送请求的，他同样也可以是只是简单的视图交互比如说弹一个对话框。mod...

展开

作者回复: 嗯，我这一讲还是单机软件，后面会谈b/s和c/s结构下的架构。

1

**诗泽**

2019-07-05

许老师可否展开讲一下如何把model 层做厚，感觉这一部分挺重要的

展开

作者回复: 假设总代码量不变，那其实就是尽可能把view和controller代码尽可能转model层

1

**吴**
2019-07-06

质量挺好

展开


**Smallfly**
2019-07-06

关于文中 MVC、MVP 的理解，跟我原先理解的不太一样，查了一些英文资料，大概是这样：

1、标准的 MVC，View 和 Model 是不能通信的，是由 Controller 监听 Model 的 DataChanged，然后去更...

展开


**Luke**
2019-07-06

最近正好在维护一个winform系统，model层的描述太棒了！
采用dom描述，主要是惯例还有业务数据一般是结构化的，个人理解


**Eason**

2019-07-06

许老师的MVC，依我理解，Facebook 的React + Redux 体现了不少。是不是以后章节也会谈到这些?

作者回复: 会谈到一些

**Geek_88604f**
2019-07-06

从分层角度，我们会倾向于认为 Model 层在最底层；View 层在中间，它持有 Model 层的 DOM 指针；Controller 层在最上方。这个分层方式有点不太理解，按我的想法view接收用户的输入它应该在最上方。麻烦许老师详细解释一下。

作者回复: 从架构的层次，不是用户响应业务流程

**大王叫我来巡山**
2019-07-05

我以前一直觉得ORM跟Model是两码事儿，但是别人都不这么认为，ORM仅仅是提供了操作数据库的一种方式，把Model做厚还是做薄是跟实际系统是有关系的，每

个系统的稳定点和变化点是不同的，范式是别人实践的总结提炼，而我们是需要根据自己遇到的问题去调整范式的。


**Aaron Cheung**
2019-07-05

打卡22 从来没有站在老师这样的高度看模型

展开


**1900**
2019-07-05

那基于数据库来实现Model层，正确的架构是什么呢?

展开

作者回复: 核心是接口是什么，不是实现


**请叫我eleven**
2019-07-05

期待老许谈谈对DDD领域驱动设计的看法。

展开

作者回复: DDD 强调的就是 Model 的使用接口要自然体现业务，不要被框架绑架。

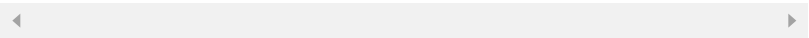
**有铭**

2019-07-05

我是一个经历过MFC时代桌面程序的后端，今天文章提到一点，让我有所感触，就是前端的Model层尽量要厚重，按我的理解，这似乎有个专用名词称呼叫“充血模型”。这一点似乎和后端流行的想法是不太一样的，后端这10年，主要逻辑都集中到Controller和Service层了，...

展开

作者回复: 后面会谈到b/s或c/s之后导致的变化



**xwhsky**

2019-07-05

光这节课就值回所有票价！

展开