理解应用架构

Gerd Wagner

2 Sep 2016

摘要

我们讨论广泛使用但不明确的 MVC 架构隐喻，并指出模型类应该与存储组件分离。

在设计复杂系统的所有情况下，无论设计新的建筑物、新的航天飞机、新的计算机或新的软件应用程序，架构都提供了一种用于定义系统结构的总体计划。任何好的建筑都是基于基本的工程原理的。**关注分离** 这有助于通过将系统分解成更小的、功能定义的部分来管理复杂性，从而使其复杂化。**相互依赖最小化** 保持更基本的部分独立于不太基本的部分。这样的架构也使得它更容易开发，或溯源，并独立于其它部分更新某些系统部分。

1 模型-视图-控制器（MVC）架构隐喻

软件应用架构中最流行和最广泛使用的方法是 Model-View-Controller 代码分区模式。虽然它没有被精确地定义，并且已经以许多不同的方式实现，特别是在 Web 应用框架中，它是基于关注点分离的原理和基本的见解，即应用程序的所有其他部分，特别是用户界面的基础是 **模型**。因此，即使 MVC 方法不能提供一个“模型”的精确定义，我们也可以认为它是一个“模型”。**基于模型** 方法。

根据维基百科，第一个 MVC 架构被Trygve Reenskaug于上世纪 70 年代引入到 SimultLoG-76 的应用程序编程中。在关于 SimalTalk 80的文章 MVC 被解释为“应用程序的三级划分”，它需要“分离（1）表示底层应用程序域模型的部分（2）将模型呈现给用户的方式，以及（3）用户与它交互的方式”。作者还使用“MVC 隐喻”一词，指出他们的方法将允许程序员“通过首先定义新的类来编写应用模型，这些新的类将体现特定的应用领域特定的信息”。

请注意，该模型被定义为包含捕获所需域信息的类。我们称之为 **模型类**。请注意，在这种早期 MVC 方法中，没有明确定义的用户界面（UI）概念。“视

图”被定义为只包含 UI 的输出端，而用户输入端与它分离，并包含在“控制器”一词之下。这并不反映 UI 是如何组织的：通过将特定形式的应用程序输出与某些形式的用户输入相结合，比如同一硬币的两面。通用 UI 概念包括输出（提供给用户的信息输出，以及系统动作）和输入（包括由用户提供的信息输入以及执行的动作）。

最初的 SimalTalk MVC 隐喻是针对（单色）基于文本屏幕的用户界面开发的，没有 UI 事件的一般概念。这可以解释为什么他们没有考虑 UI 的一个完整概念。虽然它们区分了模型中的对象状态和 UI 中的状态它们都在用户会话的范围内，但它们没有考虑模型状态和数据库状态之间的区别。

在 Martin Fowler 的网络文章图形用户界面架构（2006）中，他以以下方式总结了原始 MVC 方法的主要原则：

1. UI 与模型的分离
2. 将 UI 划分为“控制器”和“视图”
3. 视图与模型同步（通过数据绑定机制）

虽然第一和第三原则是软件应用架构的基础，但是第二个原则只是一个历史意义，很快被 SMALLATE 的开发者抛弃了。

与 20 世纪 80 年代相比，计算机、人机交互和软件应用架构已经发展。特别是，Web 作为主要的计算平台的建立使得 Web 浏览器成为用户界面最重要的基础设施。

MVC 术语在今天仍然被广泛使用，特别是 Web 应用框架，但其含义不同于“M”、“V”和“C”。通常，“视图”表示基于 HTML 表单的用户界面，而“控制器”负责在“视图”和“模型”之间进行中介，通常通过对象关系映射（ORM）方法与底层 SQL 数据库技术紧密耦合，违反了最小化相互依赖性的原则。

例如，活动记录有影响力的 Ruby on Rails 框架的范例已经被许多其他的 Web 应用框架（如 CAKEPHP）所采用，“模型”是底层数据库系统的模式的直接表示，其中数据库的每个实体表由一个“模型”类表示，该类继承了用于执行创建/检索/更新/删除（CRUD）操作的数据操作方法。在这个表中，为了建模类映射方法，“模型”依赖于底层数据库的模式，因此与底层 ORM 数据存储技术紧密耦合。虽然这可能是一种适合于数据库第一开发方法的方法，其中 SQL 数据库是应用程序的基础，但它并不是一般的方法，它将模型转换为二级资产。

此外，在基于 ORM 注释的框架中，如 JavaEE 与 JPA 注释、C# 框架 ASP.NET MVC 与实体框架和数据注释，或者 PHP 框架 SCOFRONY 与理论注释，“模型”通过 ORM 注释结合到模型代码中与底层 ORM 技术

耦合，从而使模型依赖于 ORM 技术的使用。所有这些框架都使用数据映射器基于 ORM 注释的 CRUD 操作方法。

2 洋葱架构隐喻

“洋葱架构”一词是由 Jeffry Palermo 在 2008 的一系列博客帖子中提出的。该架构隐喻的主要原则是（1）使用依赖层次结构，其中较少的基础（或中心）部分依赖于更基本的部分，但决不是相反的；（2）最基本的部分是模型，它以模型类的形式实现应用程序的数据模型，而数据存储是一个模型。分离的和基本的部分，不能与模型耦合。

事实上，Palermo 和他的追随者对这个架构隐喻做了更多的介绍，比如使用“库接口”和“服务接口”，但我认为这对洋葱隐喻并不重要。此外，他们使用的是不同的术语。当他们使用“域模型”这个术语而不是简单的模型时，他们把“域模型”和“数据模型的实现”混淆了，这是模型类所做的。从信息设计模型导出数据模型，该模型本身可以从域信息模型导出。这是基于模型的软件工程中的基本开发链。

原则上，如果数据映射方法不是基于特定于平台的 ORM（注释）技术，而是依赖于某种平台无关映射逻辑，则可以用于洋葱架构中的存储管理。

3 模型-存储-视图-控制器（MSVC）架构模式

应该清楚的是，涉及数据管理的任何软件应用程序的三个最重要的部分是：

1. 这个 **模型** 类，实现应用程序的数据模型，定义数据结构和约束；
2. 数据 **存储** 系统，通常是 SQL 数据库系统，但不一定是 SQL 数据库系统；
3. 这个 **用户界面**（UI），包括对用户的信息提供（或输出），例如，在计算机屏幕上，以及用户行为以 UI 事件的形式（例如键盘或鼠标事件）提供的用户输入，从而支持所有所需的用户交互。

MSVC 架构模式遵循洋葱架构隐喻的基本原理，通过将模型层不仅从 UI 层分离，而且从数据存储层中分离出来，并使模型成为最基本的部分，它不依赖于任何其他部分。MSVC 架构模式也遵循 MVC 架构隐喻的良好部分及其广泛使用的术语，通过使用与用户界面（UI）互换的术语视图，以及术语控制器，用于表示将 UI 代码与底层模型类和存储管理代码集成所需的胶粘代码层，或 MVC 术语，用于将视图与模型集成。

使用基于模型的方法，**通过对应用程序的数据模型进行编码来获得应用程序的模型类**。它通常以 UML 类图的形式表达。因为模型和数据存储层的任务是定义和验证约束并管理持久数据，所以我们需要一个通用的模式代码来处理这个问题，以避免用于约束验证的每个类和每个属性样板代码，以及用于数据存储管理的每类样板代码。

4 “视图模型”：区分“逻辑”与“物理”用户接口

逻辑 UI 模型的概念，也称为“视图模型”，是由 Martin Fowler 在 2004 中，在他的表示模型文章中首次提出的（在不同的名称下）。他声明视图模型“将视图的状态和行为拉到 [View] 模型类中。”这意味着 UI 的逻辑内容是从一个具体的“物理”UI 中抽象出来的，它具有逻辑 UI 字段和“命令”（用户操作）的特定渲染。逻辑 UI 字段以 UI 小部件的形式呈现，它们可以有自己的状态，而逻辑 UI 命令以适当的 UI 事件的形式呈现。

后来，在 2005，John Gossman（微软）在他的博客文章WPF 应用程序的模型/视图/视图模型介绍中采用了视图模型概念。并在架构模式首字母缩写“MVVM”下推广。

在创建/检索/更新/删除（CRUD）数据管理操作的用户界面中，**视图模型类**将绑定到一个模型类，但可以支持多个视图（类）。视图模型类将具有绑定到支持视图的窗口小部件的属性，使用单向或双向 **数据绑定** 和绑定到相应命令的方法（**命令绑定**）

通常，大多数视图字段直接对应于基础模型类的属性，尽管它们可能具有不同的名称。对于这些字段（或视图模型属性），需要对相应的模型属性进行数据绑定。但是，视图模型类也可以具有附加属性，其中一些属性可以表示不绑定到模型类属性的视图字段，而另一些可以表示 UI 中未显示的辅助字段。

视图模型类的方法在用户通过创建 UI 命令时发出相应的命令，该命令已被绑定到该命令。

将整个 UI 代码划分为视图模型部分和视图部分创建一定的开销，在某些情况下可能不合理。虽然视图模型的使用对于所有具有 CRUD 数据管理操作的应用程序都是合理的，但对于可视化用户界面来说，它可能是不合理的。

视图模型的主要优点是：（1）它们有助于 UI 设计，（2）它们有助于 UI 逻辑的测试，以及（3）它们便于 UI 的维护。