

# 基于 MVVM 模式的 WEB 前端框架的研究

易剑波

(东南大学 计算机科学与工程学院, 江苏 南京 211189)

**摘要:** MVVM 架构<sup>[1]</sup>模式是在经典的 MVC 模式<sup>[2]</sup>上发展起来的一种架构模式, 这种模式主要用于构建基于事件驱动的 UI 平台, 对于前端开发领域中数据与界面相混合的情况特别适用。笔者首先介绍了传统 MVC 架构并将它与 MVVM 架构模式做了对比研究, 然后基于目前前端开发中一个比较火的框架 Vue.js, 详细介绍了前端 MVVM 模式的工作原理, 最后详细分析了前端 MVVM 框架实现的关键技术。

**关键词:** MVC; MVVM; VUE; Web 前端

**中图分类号:** TP311.52      **文献标识码:** A      **文章编号:** 1003-9767 (2016) 19-076-03

随着 Web2.0 的发展和移动互联网时代的到来, 前端开发在整个 Web 应用软件开发中的地位越来越重要。现在的 Web 系统中越来越多的数据处理和业务逻辑开始偏向于前端, 逐渐形成了“大前端”的局面, 前端对性能和开发效率的要求也越来越高。在这种需求的推动下, jQuery 工具库、MVC 模式、MVVM 模式被相继引入到前端开发领域。其中, MVVM 模式是近一两年才开始被引入前端开发的领域, 它是基于传统 MVC 模式的进一步发展, 特别适合于在前端领域构建基于事件驱动的 UI 开发平台。

## 1 MVC 模式与 MVVM 模式

MVC 模式软件工程化设计中的一种规范, 它的思想是对软件的架构进行层次划分, 用来将数据、用户界面和业务处理逻辑相分离, 当用户界面或交互方式发生改变时, 不需要重写业务处理逻辑, 这样可以达到软件模块复用的效果。整个软件系统被分为 Model (模型)、View (视图) 和 Controller (控制器) 三部分。其中, 视图 (View) 是软件的界面; 模型 (Model) 是视图类所需要的数据, 例如表格需要显示的文字; 控制器 (Controller) 连接视图类和模型类, 其任务是使数据 (Model) 显示在视图 (View) 上。MVC 架构图如图 1 所示。

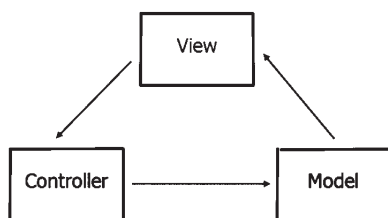


图 1 MVC 模式架构图

在 MVC 架构模式中, 各部分之间的通信过程为: View 传送指令到 Controller; Controller 完成业务逻辑后, 要求

Model 改变状态; Model 将新的数据发送到 View, 界面得到更新。在上述过程中, 每个步骤的通信都是单向的。

MVC 框架的出现, 使系统各层任务明确、逻辑清晰, 提高了代码重用性, 并降低了后期的维护成本。

MVVM 是在 MVC 模式上进一步发展的产物, 它的全称是 Model View ViewModel。该架构最初是由微软的 Martin Fowler 提出, 它的关注点在能够支持事件驱动的 UI 开发平台。MVVM 模式的核心是对 View 和 ViewModel 进行双向数据绑定, 当 ViewModel 的状态发生变化时, 这种变化可以自动传递给视图 View。MVVM 模式架构图如图 2 所示:

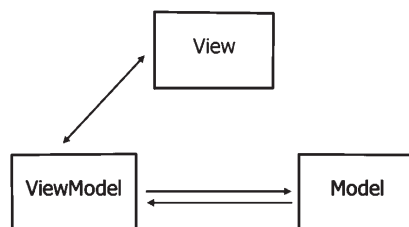


图 2 MVVM 模式架构图

MVVM 架构如图 2 所示。其中 Model 仅仅只是代表应用程序所需的数据信息, 它不关注任何行为; View 是软件中与用户进行直接交互的部分, 它需要响应 ViewModel 的事件并格式化数据, 不负责控制应用的状态; ViewModel 用于封装业务逻辑层, 这点类似于 MVC 模式中的控制器, 它控制 View 的很多显示逻辑, 它可以把数据模型的变化传递给视图, 也可以把视图中数据的变化传递给数据模型, 即在 Model 和 View 之间建立了双向绑定。

## 2 前端 MVVM 框架的运行原理

本章将以 Vue.js 为例, 讲解前端 MVVM 框架的实现原理。Vue.js 是目前前端开发领域很火的一个框架, 它是对 MVVM 架构模式的一个实现, 通过双向数据绑定连接视图层和数据

模型层,而实际的界面 UI 操作 (DOM 操作) 被封装成对应的指令 (Directives) 和过滤器 (Filters)。Vue.js 的原理如图 3 所示:

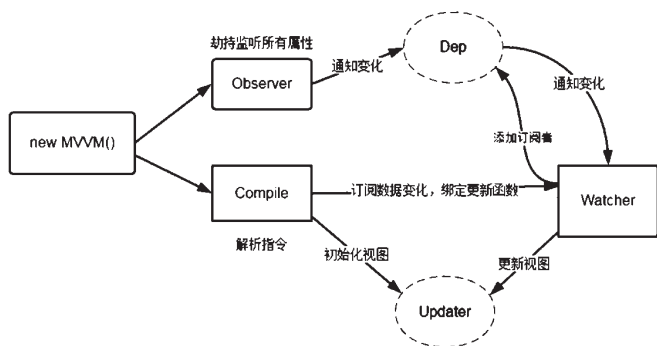


图 3 Vue.js 结构图

在 Vue.js 的架构图中, Observer 相当于观察者模式中订阅者对象的代理,它的主要作用是监听数据模型的变化,并将数据变化通知给订阅者; Compile 是一个编译器,它的作用是对视图 (Web 前端中即 HTML 元素) 中绑定的指令编译解析,根据定义动态模板将指令替换为最终展示给用户真实数据,同时它还用于绑定界面对应的更新函数; Watcher 是订阅-发布模式中订阅者的实现,它的作用是订阅数据变化的通知来执行指令所绑定的回调函数,同时 Watcher 还充当了连接监听器 Observer 和指令编译器 Compile 的桥梁; Dep 的角色相当于一个消息订阅器,它的主要作用是维护订阅者 Watcher 的信息,当系统中的数据发生变化时出发通知函数 (notify) 并调用 Watcher 的更新方法 (update) 来进行视图的更新。

当新建一个 Vue 对象时,框架进入初始化阶段。Vue 在初始化阶段主要执行两个操作:一是遍历系统中数据的所有属性,来对各个属性的变化添加监听;第二个操作是利用指令编译器 Compile 对视图中绑定的指令进行扫描进行视图的初始化,然后订阅 Watcher 来更新视图,此时 Watcher 会将自己添加到消息订阅器 Dep 中。至此,Vue 的初始化过程结束。

在系统运行过程中,一旦系统中的数据模型发生了变化,观察者 Observer 的 setter 访问器属性就会被触发,此时消息订阅中心 Dep 会遍历它所维护的所有订阅者,对于每一个订阅了该数据的对象,向它发出一个更新通知,订阅者收到通知后就会对视图进行相应的更新。以上过程不断往复循环,这就是 MVVM 模式在 Vue.js 中的运行原理。

### 3 前端 MVVM 框架实现的关键技术

前一章节中以目前比较火的框架 Vue.js 为例介绍了前端领域中 MVVM 模式的实现原理,本章将针对上述原理图,深入分析前端 MVVM 框架实现的关键技术。主要包含以下几点。

#### (1) 数据变化的监听。

在 Web 前端中,数据变化的监听主要通过对象的访问器属性来实现。访问器属性是通过 Object.defineProperty() 方法

来实现,它不能直接在对象中进行设置。具体使用方法如下:

```

var o={};
Object.defineProperty(o, 'propA', {
  get: function() {
    // return something ...
  },
  set: function(value) {
    // do something here ...
  }
});

```

通过上述方法即给对象 o 设置了访问器属性,当使用 o.propA 语句获取对象 o 的 propA 属性时,内部的 getter 就会被自动调用;当使用 o.propA=someValue 来对对象 o 的 propA 属性进行赋值时,内部的 setter 访问器就会被自动调用。可以在 setter 和 getter 访问器属性被自动调用时来进行一些逻辑处理,这样就实现了对数据变化的监听。

#### (2) 双向绑定的实现。

双向绑定即将视图 View 和数据 Model 进行相互关联,使一端的变化能同步反映在另一端。在前端中,这主要通过上一小节中介绍的对象的访问器属性和 HTML 元素的事件监听来实现。

例如,对于一个输入框视图 <input type='text' id='myText' />,要将其与数据对象 o 的 propA 属性进行双向数据绑定。则可以按照如下的步骤来完成:

```

var o={};
Object.defineProperty(o, 'propA', {
  set: function(newValue) {
    document.getElementById( ' myText ' ).value=newValue;
    // 在 Model 发生变化时,更新对应的视图
  }
});

document.getElementById( ' myText ' ).addEventListener(
  ' keyup ', function(e) {
    o.propA=e.target.value; // 监听 View 的变化,同步更新 Model
  });

```

#### (3) 发布-订阅模式<sup>[3]</sup>的应用。

上述介绍了简单的一对一双向绑定的实现,即一个数据模型只与一个视图进行绑定。当多个 View 与一个 Model 进行绑定时,每次更新 Model 时需要在 Model 的 set 访问器属性中更新多个 View,这样硬编码的方式不利于后期的维护。为了解决硬编码带来的耦合性过强的问题,在实际实现中,需要使用到设计模式中的发布-订阅模式。

发布-订阅模式 (又称观察者模式) 是一种常用的设计模式,该模式包含发布者和订阅者两种角色。可以让多个订

(下转第 84 页)

算法也逐渐显示出这类似的情况。

(4) 数组为 1 000 000 个元素时的情况, 如图 4 所示。

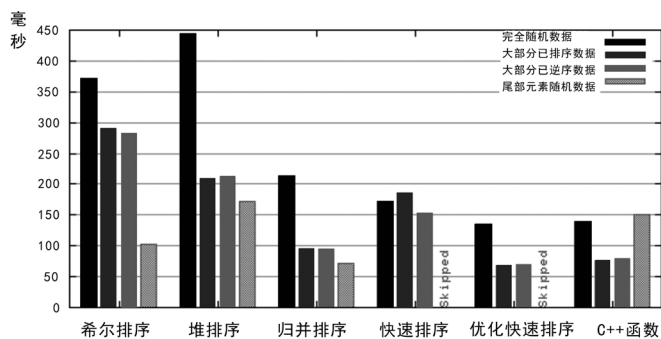


图 4 数组为 1 000 000 个 Integers 元素的排序

当数据规模增至 1 000 000 个元素时, 同样是“尾部元素随机数据”测试用例对于优化快速排序算法的病态状况已更加明显地呈现, 它和普通快速排序一样, 其测试值被忽略。如图 4 所示, 当数组长度增加时, 与普通快速排序相比, 优化快速排序的效率提升也更加明显。而且, 归并排序在此情况下表现较好。

## 参考文献

- [1] 陈思敏. 基于 C 语言的几种排序算法的分析 [J]. 电子设计工程, 2013, 21(17).
- [2] 张乃孝, 陈光, 孙猛. 算法与数据结构 [M]. 北京: 高等教育出版社, 2002.

(上接第 77 页)

读者订阅同一个发布者发布的主题, 当发布者的主题发生变化时, 对外发送一个通知, 所有订阅了该主题的订阅者都会接收到更新的消息。因此, 观察者模式定义的是一种一对多的关系。

可以看到, 发布-订阅模式非常适合于 MVVM 双向绑定中多个视图绑定到同一个数据模型的情形。通过将视图看作一个观察者对象, 如果该视图与某个数据进行了双向绑定, 则将其加入到数据模型的订阅列表中, 当数据变化时, 这些视图就能得到通知以进行更新。

(4) 使用 DocumentFragment<sup>[4]</sup> 优化性能。

在某些情况下, 当数据发生改变时可能需要动态生成界面元素 (HTML 标签) 并插入到视图对应的位置。如果生成的数据量较大时 (例如根据数据生成一个很大的列表, 然后将列表逐条插入到页面中), 会引起页面多次重绘而影响性能。这种情况下, 可以使用 DOM (Document Object Model) 中的文档片段对象即 DocumentFragment 来提升性能。

DocumentFragment 即文档片段, 在片段其中可以含有多个子节点, 因此文档片段的作用相当于是一个节点容器。在将文档片段插入到 HTML 页面中时, 只有它的子节点会被插入目标节点。由于文档片段中的节点操作都是在内存中执行完毕后一次性插入到 DOM 中, 而不是逐条插入 DOM, 因此它的性能和速度比直接操作其中包含的子节点要有很大提升, 这是 Web 前端性能优化很重要的一种方法。因此, 当多

个视图元素被插入到界面中的同一位置时, 可以先将它们都挂载到一个文档片段, 经过处理后, 将整个文档片段返回插入挂载目标。

## 4 结 语

随着前端的快速发展, 无论是用户界面还是业务处理逻辑都变得越来越复杂, 使用原生 JavaScript 或 jQuery 去操控 DOM 变得越来越不现实。MVVM 模式的出现, 让开发者只需要控制一个 ViewModel 并结合动态模板来显示界面, 复杂的 Web 应用程序可以在不需要传统的元素选择器的情况下做到简单可维护。框架的学习固然有一定的成本, 但是当 Web 应用的功能和复杂度达到一定的程度时, 使用优秀的 MVVM 框架反而在总体上会更节约成本。

## 参考文献

- [1] 刘立. MVVM 模式分析与应用 [J]. 微型电脑应用, 2012, 28(12): 57-60.
- [2] 许鑫, 费翔林. 基于 MVC 模式的应用软件开发框架研究 [J]. 计算机工程与应用, 2005, 41(30): 102-104.
- [3] 张宁, 王越, 王东. 观察者模式及其在软件开发中的应用 [J]. 大众科技, 2008(11): 35-36.
- [4] 康长安, 陈玉红. 基于前端的 Web 性能优化 [J]. 电脑知识与技术, 2011, 7(16): 3811-3813.

word版下载: <http://www.ixueshu.com>

免费论文查重: <http://www.paperyy.com>

3亿免费文献下载: <http://www.ixueshu.com>

超值论文自动降重: [http://www.paperyy.com/reduce\\_repetition](http://www.paperyy.com/reduce_repetition)

PPT免费模版下载: <http://ppt.ixueshu.com>

---

## 阅读此文的还阅读了:

- [1. 基于STRUTS框架开发MVC模式的WEB应用](#)
- [2. 基于MVVM构架的Web前端框架研究](#)
- [3. 基于MVC模式的Struts框架技术的研究与应用](#)
- [4. 基于CS架构的MVVM框架的改进研究](#)
- [5. 基于Bootstrap响应式Web前端研究](#)
- [6. 基于MVC模式的Struts2框架在Web应用开发中的实现](#)
- [7. 基于MVVM模式的WEB前端框架的研究](#)
- [8. 基于Jersey框架的RESTful Web Service的研究与实现](#)
- [9. Web前端MVC框架的意义分析](#)
- [10. 初探WEB前端技术](#)
- [11. MVVM设计模式及其应用研究](#)
- [12. 基于BootStrap 的WEB前端开发应用研究](#)
- [13. 基于MVC设计模式Struts框架结构的研究](#)
- [14. 基于MVC模式的Web应用框架研究](#)
- [15. 基于bootstrap响应式web前端分析](#)
- [16. 基于Velocity的可配置前端框架重构](#)
- [17. 基于HTML5的前端效率机制及Web应用模式研究](#)
- [18. Web前端中MVVM框架的应用研究](#)
- [19. 基于路由模式的前端框架设计与改进](#)
- [20. 基于VueJs的WEB前端开发研究](#)
- [21. Web前端开发技术的研究](#)
- [22. 基于MVC设计模式的WEB应用框架研究](#)
- [23. Twitter Bootstrap:前端框架利器](#)
- [24. 基于MVC模式的struts框架的研究与应用](#)
- [25. MiniUI前端框架研究](#)

- [26. 一种基于AJAX的Web应用框架的研究与设计](#)
- [27. 基于jQuery前端框架提升Web用户体验的研究](#)
- [28. MVVM 设计模式及其应用研究磁](#)
- [29. 基于MVC模式的Struts框架的研究](#)
- [30. MVVM模式分析与应用](#)
- [31. 基于CM的MVVM设计模式应用研究](#)
- [32. 基于框架的web前端\(Bootstrap和MUI\)之比较](#)
- [33. Web前端MVC框架的意义研究](#)
- [34. 基于Web服务的股本交易系统框架的研究](#)
- [35. 基于MVC模式的Struts框架在WebGIS模型中的研究](#)
- [36. MVVM框架在Web前端的应用研究](#)
- [37. 前端——开源前端框架纵横谈](#)
- [38. 基于Spring+Hibernate框架的Web应用系统的研究](#)
- [39. 浅谈基于HTML5的Web前端APP开发](#)
- [40. 基于Vue.js的Web前端应用研究](#)
- [41. 基于MVVM构架的Web前端框架研究](#)
- [42. 一种基于MVC模式Web开发框架WebWork的研究](#)
- [43. 基于MVC模式的Struts框架的研究](#)
- [44. 基于MVC和MVVM的天地图架构模式研究](#)
- [45. 基于MVVM架构的移动Web前端展示方案](#)
- [46. Silverlight下的MVVM模式的应用](#)
- [47. 基于struts框架的WEB应用](#)
- [48. Web前端开发技术的研究](#)
- [49. 基于MVC设计模式的SSH框架的研究](#)
- [50. MVVM框架在Web前端的应用研究](#)