

## 26 | 实战(上)：怎么设计一个“画图”程序？

2019-07-19 许式伟

许式伟的架构课

[进入课程 >](#)



讲述：姚迪迈

时长 11:03 大小 10.14M



你好，我是七牛云许式伟。

到上一讲为止，桌面程序架构设计的基本结构就讲完了。直到现在为止，我们没有讨论任何与具体的应用业务逻辑本身相关的内容。这是因为探讨的内容是普适有效的设计理念，整个讨论会显得很抽象。

今天我们结合一个实际的应用案例，来回顾一下前面我们介绍的内容。

我们选择了做一个“画图”程序。选它主要的原因是画图程序比较常见，需求上不需要花费过多的时间来陈述。

我们前面说过，一个 B/S 结构的 Web 程序，基本上分下面几块内容。

Model 层：一个多用户 ( Multi-User ) 的 Model 层，和单租户的 Session-based Model。从服务端来说，Session-based Model 是一个很简单的转译层。但是从浏览器端来说，Session-based Model 是一个完整的单租户 DOM 模型。

View 层：实际是 ViewModel 层，真正的 View 层被浏览器实现了。ViewModel 只有 View 层的数据和可被委托的事件。

Controller 层：由多个相互解耦的 Controller 构成。切记不要让 Controller 之间相互知道对方，更不要让 View 知道某个具体的 Controller 存在。


画图程序的源代码可以在 Github 上下载，地址如下：

<https://github.com/qiniu/qpaint>

今天我们讨论浏览器端的 Model，View 和 Controller。

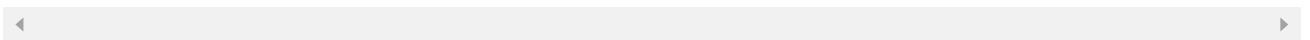
## Model 层

我们先看 Model 层。浏览器端的 Model 层，代码就是一个 [dom.js](#) 文件。它是一棵 DOM 树，根节点为 QPaintDoc 类。整个 DOM 树的规格如下：

 复制代码

```
1 class QLineStyle {
2   properties:
3     width: number
4     color: string
5   methods:
6     constructor(width: number, color: string)
7 }
8
9 class QLine {
10  properties:
11    pt1, pt2: Points
12    lineStyle: QLineStyle
13  methods:
14    constructor(pt1, pt2: Point, lineStyle: QLineStyle)
15    onpaint(ctx: CanvasRenderingContext2D): void
16 }
17
18 class QRect {
19  properties:
20    x, y, width, height: number
21    lineStyle: QLineStyle
22  methods:
```

```
23     constructor(r: Rect, lineStyle: QLineStyle)
24     onpaint(ctx: CanvasRenderingContext2D): void
25 }
26
27 class QEllipse {
28   properties:
29     x, y, radiusX, radiusY: number
30     lineStyle: QLineStyle
31   methods:
32     constructor(x, y, radiusX, radiusY: number, lineStyle: QLineStyle)
33     onpaint(ctx: CanvasRenderingContext2D): void
34 }
35
36 class QPath {
37   properties:
38     points: []Point
39     close: bool
40     lineStyle: QLineStyle
41   methods:
42     constructor(points: []Point, close: bool, lineStyle: QLineStyle)
43     onpaint(ctx: CanvasRenderingContext2D): void
44 }
45
46 interface Shape {
47   onpaint(ctx: CanvasRenderingContext2D): void
48 }
49
50 class QPaintDoc {
51   methods:
52     addShape(shape: Shape): void
53     onpaint(ctx: CanvasRenderingContext2D): void
54 }
```



目前这个 DOM 还是单机版本的，没有和服务端的 Session-based Model 连起来。关于怎么连，我们下一讲再讨论。

这个 Model 层的使用是非常容易理解的，也非常直观体现了业务。主要支持的能力有以下两个方面。

其一，添加图形（Shape），可以是 QLine，QRect，QEllipse，QPath 等等。

其二，绘制（onpaint）。前面我们介绍 MVC 的时候，我曾提到为了 View 层能够绘制，需要让 DOM 层把自己的数据暴露给 View 层。

但是从简洁的方式来说，是让 Model 层自己来绘制，这样就避免暴露 DOM 层的实现细节。虽然这样让 Model 层变得有那么一点点不纯粹，因为和 GDI 耦合了。但是我个人认为耦合 GDI 比暴露 DOM 的数据细节要好，因为 GDI 的接口通常来说更稳定。

依赖选择是考虑耦合的一个关键因素。在依赖选择上，我们会更倾向于依赖接口更为稳定的组件，因为这意味着我们的接口也更稳定。


## ViewModel 层

我们再看 ViewModel 层。它的代码主要是一个 [index.htm](#) 文件和一个 [view.js](#) 文件。index.htm 是总控文件，主要包含两个东西：

界面布局（Layout）；

应用初始化（InitApplication），比如加载哪些 Controllers。

而 [view.js](#) 是我们 ViewModel 层的核心，实现了 QPaintView 类。它的规格如下：

 复制代码

```
1 interface Controller {
2   stop(): void
3   onpaint(ctx: CanvasRenderingContext2D): void
4 }
5
6 class QPaintView {
7   properties:
8     doc: QPaintDoc
9     properties: {
10       lineWidth: number
11       lineColor: string
12     }
13   drawing: DOMElement
14   controllers: map[string]Controller
15   methods:
16     get currentKey: string
17     get lineStyle: QLineStyle
18     onpaint(ctx: CanvasRenderingContext2D): void
19     invalidateRect(rect: Rect): void
20     registerController(name: string, controller: Controller): void
21     invokeController(name: string): void
22     stopController(): void
23     getMousePos(event: DOMEvent): Point
24   events:
25     onmousedown: (event: DOMEvent):void
```

```
26  onmousemove: (event: DOMEvent):void
27  onmouseup: (event: DOMEvent):void
28  ondblclick: (event: DOMEvent):void
29  onkeydown: (event: DOMEvent):void
30  }
31
32  var qview = new QPaintView()
```

看起来 QPaintView 的内容有点多，我们归类一下：

**和 Model 层相关的，就只有 doc: QPaintDoc 这个成员。有了它就可以操作 Model 层了。**

**属于 ViewModel 层自身的，数据上只有 properties 和 drawing。**其中 properties 是典型的 ViewModel 数据，用来表示当前用户选择的 lineWidth 和 lineColor 等。drawing 则是浏览器对 HTML 元素的抽象，通过它以及 JavaScript 全局的 document 对象就可以操作 HTML DOM 了。

当然 ViewModel 层一个很重要的责任是绘制。onpaint 和 invalidRect 都是绘制相关。invalidRect 是让界面的某个区域重新绘制。当前为了实现简单，我们总是整个 View 全部重新绘制。

前面我说过，Web 开发一个很重要的优势是不用自己处理局部更新问题，为什么这里我们却又要自己处理呢？原因是我们没有用浏览器的 Virtual View，整个 DOM 的数据组织完全自己管理，这样我们面临的问题就和传统桌面开发完全一致。

剩下来的就是 Controller 相关的了。主要功能有：

registerController（登记一个 Controller），invokeController（激活一个 Controller 成为当前 Controller），stopController（停止当前 Controller），View 层并不关心具体的 Controller 都有些什么，但是会对它们的行为规则进行定义；

事件委托（delegate），允许 Controller 选择自己感兴趣的事件进行响应；

getMousePos 只是一个辅助方法，用来获取鼠标事件中的鼠标位置。

View 层在 MVC 里面是承上启下的桥梁作用。所以 View 层的边界设定非常关键。

如果我们把实际绘制 ( onpaint ) 的工作交给 Model 层，那么 View 基本上就只是胶水层了。但是就算如此，View 层仍然承担了一些极其重要的责任。

屏蔽平台的差异。Model 层很容易做到平台无关，除了 GDI 会略微费劲一点；Controller 层除了有少量的界面需要处理平台差异外，大部分代码都是响应事件处理业务逻辑，只要 View 对事件的抽象得当，也是跨平台的。

定义界面布局。不同尺寸的设备，界面交互也会不太一样，在 View 层来控制不同设备的整体界面布局比较妥当。

## Controller 层

最后我们看下 Controller 层。Controller 层的文件有很多，这还有一些 Controller 因为实现相近被合并到一个文件。详细信息如下。

Menu, PropSelectors, MousePosTracker : [accel/menu.js](#)

Create Path : [creator/path.js](#)

Create FreePath : [creator/freepath.js](#)

Create Line, Rect, Ellipse, Circle : [creator/rect.js](#)

其中，[menu.js](#) 主要涉及各种命令菜单和状态显示用途的界面元素。用于创建各类图形 ( Shape )，选择当前 lineWidth、lineColor，以及显示鼠标当前位置。

在创建图形这些菜单项上，有两点需要注意。

其一，菜单并不直接和各类创建图形的 Controller 打交道，而是调用 qview.invokeController 来激活对应的 Controller，这就避免了两类 Controller 相互耦合。


其二，虽然前面 Model 层支持的图形只有 QLine、QRect、QEllipse、QPath 等四种，但是界面表现有六种：Line、Rect、Ellipse、Circle、Path、FreePath 等等。这是非常正常的现象。同一个 DOM API 在 Controller 层往往会有多条实现路径。

选择当前 lineWidth、lineColor 操作的对象是 ViewModel 的数据，不是 Model。这一点前面几讲我们也有过交代。我们当时举的例子是 Selection。其实你把当前 lineWith、

lineColor 看作是某种意义上的 Selection，也是完全正确的认知。

鼠标位置跟踪（MousePosTracker）是一个极其简单，但也是一个很特殊的 Controller，它并不操作任何正统意义的数据（Model 或 ViewModel），而是操作输入的事件。

剩下的几个 JavaScript 文件都是创建某种图形。它们的工作机理非常相似，我们可以随意选一个看一下。比如 QRectCreator 类，它的规格如下：

 复制代码

```
1 class QRectCreator {
2   methods:
3     constructor(shapeType: string)
4     stop(): void
5     onpaint(ctx: CanvasRenderingContext2D): void
6     onmousedown: (event: DOMEvent):void
7     onmousemove: (event: DOMEvent):void
8     onmouseup: (event: DOMEvent):void
9     onkeydown: (event: DOMEvent):void
10 }
```

在初始化（构造）时，QRectCreator 要求传入一个 shapeType。这是因为 QRectCreator 实际上并不只是用于创建 Rect 图形，还支持 Line、Ellipse、Circle。只要通过选择两个 points 来构建的图形，都可以用 QRectCreator 这个 Controller 来做。

QRectCreator 接管了 View 委托的 mousedown、mousemove、mouseup、keydown 事件。

其中，mousedown 事件记录下第一个 point，并由此开启了图形所需数据的收集过程，mouseup 收集第二个 point，随后后创建相应的 Shape 并加入到 DOM 中。keydown 做什么？它用来支持按 ESC 放弃创建图形的过程。

## 架构思维上我们学习到什么？

通过分析这个“画图”程序，你对此最大的收获是什么？欢迎留言就此问题进行交流。这里我也说说我自己想强调的点。

首先，这个程序没有依赖任何第三方库，是裸写的 JavaScript 代码。关于这一点，我想强调的是：

第一，这并不是去鼓励裸写 JavaScript 代码，这只是为了消除不同人的喜好差异，避免因不熟悉某个库而导致难以理解代码的逻辑；

第二，大家写代码的时候，不要被框架绑架，框架不应该增加代码的耦合，否则这样的框架就应该丢了；更真实的情况是，你很可能是在用一个好框架，但是是不是真用好了，还是取决于你自己的思维。

从架构设计角度来说，在完成需求分析之后，我们就进入了架构的第二步：概要设计（或者也可以叫系统设计）。这个阶段的核心话题是分解子系统，我们关心的问题是下面这些。

每个子系统负责什么事情？

它依赖哪些子系统？它能够少知道一些子系统的存在么？

它们是通过什么接口耦合的？这个接口是否自然体现了两者的业务关系？它们之间的接口是否足够稳定？

MVC 是一个分解子系统的基本框架，它对于桌面程序尤为适用。通过今天对“画图”程序的解剖，我们基本能够建立桌面程序框架上非常一致的套路：

Model 层接口要自然体现业务逻辑；

View 层连接 Model 与 Controller，它提供事件委托（delegate）方便 Controller 接收感兴趣的事件，但它不应该知道任何具体的 Controller；

Controller 层中，每个 Controller 都彼此独立，一个 Controller 的职责基本上就是响应事件，然后调用 Model 或 ViewModel 的接口修改数据。

当然，这里没有讨论特定应用领域本身相关的架构问题。对于桌面程序而言，这件事通常发生在 Model 层。但对于我们今天的例子“画图”程序而言，Model 层比较简单，基本上还不太需要讨论。在后面，我们也可能会尝试把这个“画图”程序需求变复杂，看架构上应该怎么进行应对。

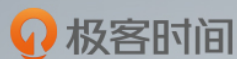
## 结语



今天我们结合一个大家非常熟悉的例子“画图”程序来介绍 MVC 架构。虽然我们基于 Web 开发，但是我们当前给出的画图程序本质上还是单机版的。

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。下一讲我们将继续实战一个联网版本的画图程序。

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。



# 许式伟的架构课

从源头出发，带你重新理解架构设计

许式伟  
七牛云 CEO



新版升级：点击「👤请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 25 | 桌面开发的未来

## 精选留言 (7)

💬 写留言



八哥

2019-07-19

CEO，js写的还这么好。厉害

展开 ▾



👍 1

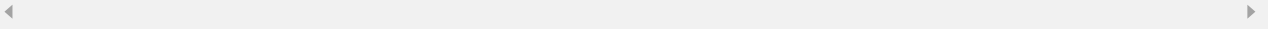


**Linuxer**

2019-07-19

由Controller来创建具体Shape这样会不会，model和controller又耦合了呢

作者回复: controller本来就会耦合model和view。重点是controller之间不要耦合



**kyle**

2019-07-19

掉队了

展开 ∨

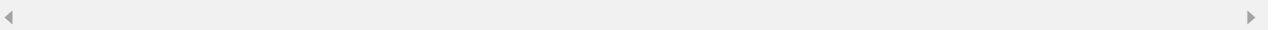


**honnkyou**

2019-07-21

老师，Model层跟view层中都又onpaint，两个区别或者侧重是什么呢？

作者回复: 我文章中有说。view层不想知道model层数据结构细节的话，最简单的方法就是把onpaint事件委托给model层。



**Luke**

2019-07-19

保持跟进，目前只能说好像有点懂了，还是得在自己项目上多思考下，不是真的懂了



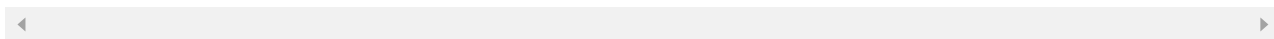
**Linuxer**

2019-07-19

QPath 这些是不是要实现interface Shape ？

展开 ∨

作者回复: 实现了



**Aaron Cheung**

2019-07-19

大前端 还是非常有价值 打卡

展开 ∨

