

24 | 跨平台与 Web 开发的建议

2019-07-12 许式伟

许式伟的架构课

[进入课程 >](#)



讲述：姚迪迈

时长 08:44 大小 8.00M



你好，我是七牛云许式伟。

上一讲我们聊了浏览器，以及移动浏览器之争：小程序与 PWA。

当我们思考浏览器从技术上带来了什么的时候，我们可以把它分为两点。

跨平台桌面程序开发；

Web 开发（B/S 架构的新型应用）。

今天我们分别就跨平台桌面程序和 Web 开发展开来聊一聊。

跨平台桌面程序开发

跨平台的桌面程序开发是一个超级难题。无数人前仆后继，各种方案层出不穷，但至今为止，仍然没有称得上真正深入人心的解决方案。

原因很简单，因为桌面程序本身的范畴在变。有两个关键的因素会导致桌面开发产生巨大的差异性。

一个因素自然是操作系统。不同的操作系统抽象的界面程序框架并不一致。这些不一致必然导致开发工作量的增加。

放弃某个操作系统，就意味着放弃某个流量入口，也就意味着放弃这些用户。所以虽然很麻烦，我们还是不得不支持着每一个主流的操作系统。

另一个因素是屏幕尺寸。就算相同的操作系统，在不同尺寸的屏幕上，交互的范式也会存在很大的差异性，这也会导致不低的跨平台工作量。

首先我们看下操作系统。

PC 本地：Windows，macOS，Linux 等等；

PC Web：Chrome，Safari，FireFox 等等；

Mobile 本地：Android，iOS 等等；

Moble Web：小程序，PWA 等等。

我们再看下屏幕尺寸。

大屏：PC、笔记本，Pad 等等；

中屏：手机；

小屏：手表。

如此繁复多样的终端类型，无怪跨平台如此之难。我们来总结一下当前都有哪些跨平台的解决方案。

方案	语言	操作系统	简介
QT	C++	Windows, macOS, Linux Android, iOS, WinRT Embedded Linux, etc.	PC 时代出现的跨平台方案，但是今天仍然还比较活跃，支持 Web 之外的主流操作系统。
wxWidgets	C++	Windows, macOS, Linux	PC 时代出现的跨平台方案。
SWT, Swing	Java	Windows, macOS, Linux	PC 时代出现的跨平台方案。
kotlin-native	Kotlin	Windows, macOS, Linux WebAssembly Android, iOS	支持 Web 之外的主流操作系统。对 Web 支持主要是 WebAssembly。
React Native	JavaScript	Windows, macOS, Linux Android, iOS Web	基于 Web 技术的跨平台方案。
weex	JavaScript	Android, iOS Web	基于 Web 技术的跨平台方案。
小程序, PWA	JavaScript	Android, iOS, Web	基于 Web 技术的跨平台方案。
Flutter	Dart	Windows, macOS, Linux Android, iOS Web	基于最基础的 Canvas 实现的跨平台方案，平台依赖最小化，当然也意味着方案的工作量非常大。
PhoneGap, Cordova	JavaScript	Android, iOS	基于 Web 技术的跨平台方案。

这个列表只是沧海一粟。之所以没有列那么多，也是因为大部分的跨平台框架都已经不怎么活跃，已经无疾而终了。

目前来说，还很难说哪个方案会胜出。

关于跨平台开发，我觉得有一句话特别深刻：“每一次统一的努力，都最终变成新的分裂”。当然，这样的事情在很多领域都会发生，只是跨平台开发更加如此。

但是无论如何，跨平台的梦还会继续。

Web 开发

聊完了跨平台，我们来聊聊浏览器带来的另一面：Web 开发。

Web 的 B/S 架构意味着编写软件有了更高的复杂性。这主要表现在以下几个方面。

其一，多用户。有了 Server 端，意味着用户的数据不再是保存在 Client (Browser) 端，而是存储在 Server 端。

其二，更高的数据可靠性要求。数据在 Client 端，客户自己对数据的可靠性负责。硬盘坏了，数据丢了，用户会后悔没有对数据进行备份。

但是一旦数据在 Server 端，数据可靠性的责任方就到了软件厂商这边。如果厂商不小心把数据搞丢了，用户就会跳起来。

其三，更多可能的分工安排。详细来说，Web 应用从流派来说，分为两大类：胖前端与胖后端。

所谓胖前端，是指把尽可能多的业务逻辑放在前端。极端情况下，整个网站就是一个单页的应用。胖前端无论开发体验还是用户体验，都更接近于本地应用 (Native App)。

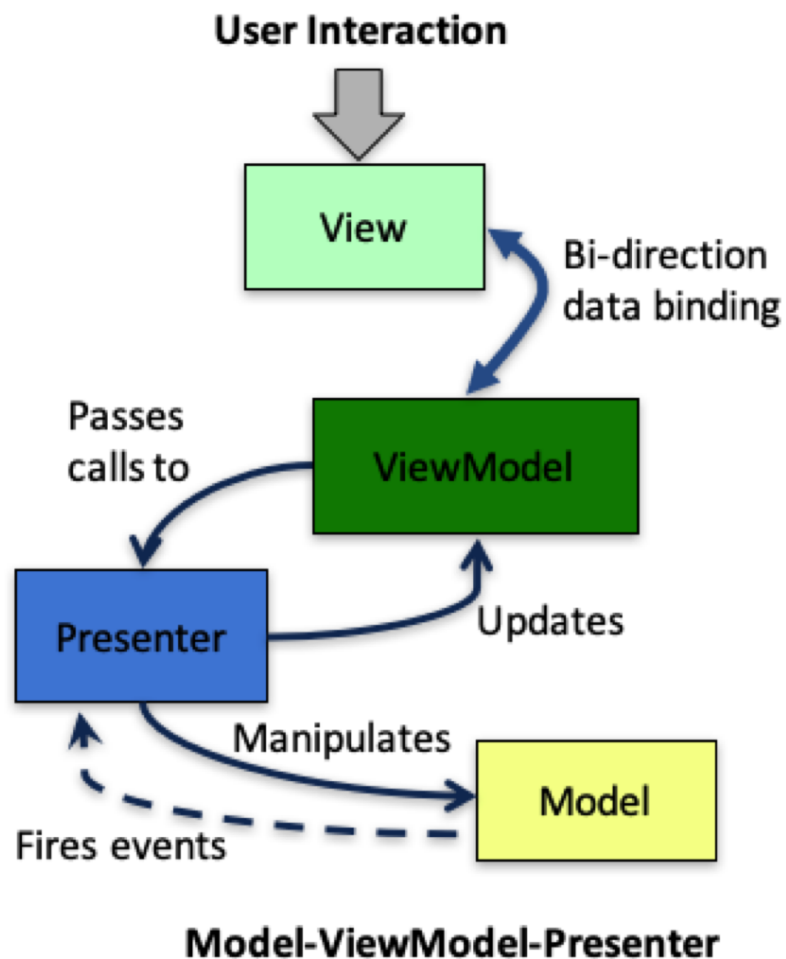
所谓胖后端，是指主要逻辑都在后端，包括界面交互的事件响应，也通过网络调用交给了后端来实现。

我们先看客户端 (Client)，也就是浏览器端 (Browser)。上一讲我们提到，浏览器的界面框架并没有窗口系统，它通过 HTML+CSS 来描述界面。

HTML+CSS 与其理解为 View 层，不如理解为 ViewModel 层，因为 HTML DOM 从数据角度完整描述了界面的样子。而 View 层已经被浏览器自己实现了。

这极大简化了界面开发的复杂性，因为界面的局部更新是一个复杂的话题，今天浏览器通过引入 HTML+CSS 这样的 ViewModel 层把它解决了。

这个时候我们重新看 MVC 框架在浏览器下的样子，你会发现它变成了 MVMP 模式，全称为 “Model-ViewModel-Presenter”。



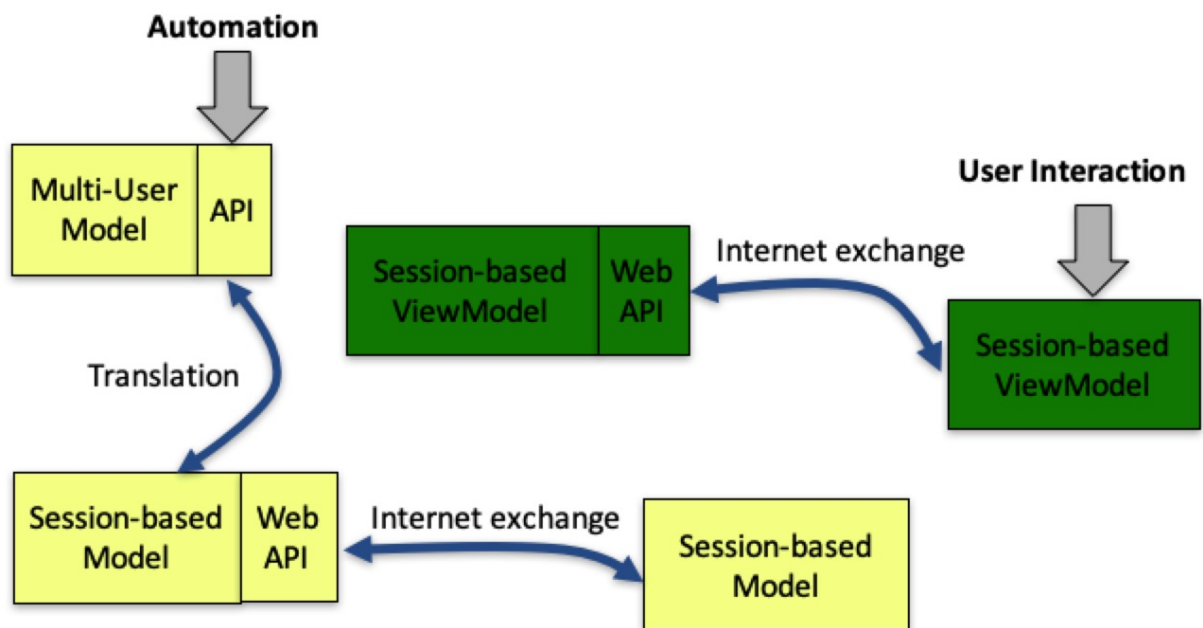
首先，我们看事件响应过程。浏览器的 View 收到了用户的交互事件，它把这些事件委托（delegate）给了 ViewModel 层，并且通过 HTML DOM 暴露出来。通过修改 HTML 元素的事件响应属性，一般名字叫 onXXX（比如 onclick），可以获得事件的响应机会。

然后我们看 Model 层的数据变化（DataChanged）事件。在标准的 MVC 模式中，Model 层的数据变化是通知到 View 层，但是在浏览器下 View 是由浏览器实现的，要想让它接受 DataChanged 事件并且去处理是不太可能了。

所以解决思路自然是让 Controller 层来做，这样就变成了 MVP 模式。但是我们又不是标准的 MVP，因为 Presenter 层更新界面（Update View）并不是操作 View，而是 ViewModel。

综上，浏览器下的 MVC，最终实际上是 MVMP（Model-ViewModel-Presenter）。

聊完了浏览器端，我们在来看下服务端（Server）。虽然这一章我们的重点不是聊服务端，但是为了有个完整的认识，我们还是要概要地梳理一下 Server 端的架构。



前面我们在“[22 | 桌面程序的架构建议](#)”中，曾提到桌面软件除了要支持用户交互外，另一个很重要的需求是提供应用程序的二次开发接口（API）。

到了 Web 开发，我们同样需要二次开发接口，只不过这个二次开发接口不再是在 Client 端完成的，而是在 Server 端完成。Server 端支持直接的 API 调用，以支持自动化（Automation）方面的需求。

所以，对 Server 端来说，最底层的是一个多租户的 Model 层（Multi-User Model），它实现了自动化（Automation）所需的 API。

在 Multi-User Model 层之上，有一个 Web 层。Web 层和 Model 层的假设不同，Web 层是基于会话的（Session-based），因为它负责用户的接入，每个用户登录后，会形成一个个会话（Session）。

如果我们对 Web 层细究的话，又分为 Model 层和 ViewModel 层。为了区分，Web 这边的 Model 层我们叫它 Session-based Model。相应地，ViewModel 层我们叫它 Session-based ViewModel。

在服务端，Session-based Model 和 Session-based ViewModel 并不发生直接关联，它们通过自己网络遥控浏览器这一侧的 Model 和 ViewModel，从而响应用户的交互。

Session-based Model 是什么样的呢？它其实是 Multi-User Model 层的转译。把多租户的 API 转译成单租户的场景。所以这一层并不需要太多的代码，甚至理论上自动实现也是有可能的。

Session-based ViewModel 是一些 HTML+JavaScript+CSS 文件。它是真正的 Web 业务入口。它通过互联网把自己的数据返回给浏览器，浏览器基于 ViewModel 渲染出 View，这样整个系统就运转起来了。

结语

今天我们聊了 Web 带来的两个重要改变。一个是跨平台，一个是 Web 开发，即 B/S 架构下的新型应用到底应该怎么实现。

从跨平台来说，这个话题是桌面程序员（也叫“大前端”）永远的痛。计划赶不上变化，用来形容大前端程序员面临的窘境是一点都不过分的。一个玩意还没搞熟悉了，另一个东西又出来了，变化太快，要跟上实属不易。

从 Web 开发来说，MVC 变成了 MVMP（Model-ViewModel-Presenter）。我们和单机的桌面软件一样的建议，认真对待 Model 层，认真思考它的使用接口是什么样的，把 Model 层做厚。

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。下一讲我们将结合一个实际的案例，来讲解一下桌面开发（含单机软件和 Web）到底是什么样的。


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 23 | Web开发: 浏览器、小程序与PWA

精选留言 (7)

 写留言



lckfa李钊
2019-07-13

因为工作的原因, 莫名其妙的成为了大前端的一员, 从Windows原生直绘界面到基于Qt的Web混合开发, 再到Flutter移动跨平台, 一路走来是越来越惊讶, 前端的知识体系气泡越吹越大, 突发学海无涯之感。不过我也一直在思考那么不变的东西, 从框架的架构角度理解, 它们其实是借鉴和传承的, 因此不论是使用C++还是JS或者Dart, 貌似都不再是难题了。学习架构课不一定是奔着学完做架构师去的, 也是为了更好的把自己的知识体系...
展开

作者回复: 



 3

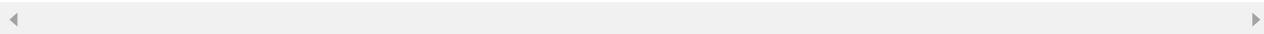


Linuxer

2019-07-12

理解起来还是有些吃力，请问能结合一下具体项目吗？最好是某个开源项目

作者回复: 开源项目可能过大，我下一讲是打算做一个小例子



👍 2



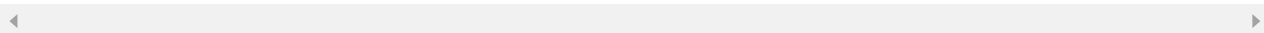
杨洪林

2019-07-12

server 架构下的automation 具体指那些动作？

展开 ▾

作者回复: 这个是指外部系统对我们业务的api请求。我们的业务提供可视化的界面提供只是一种可能性，更重要的其实是如何让更多人使用我们的核心业务。



👍 1



黄强

2019-07-12

认真对待Model层，将其做厚，相应的前端会更薄，更能适应变化的可能，再次体现架构师对稳定点，变化点的抉择

展开 ▾

作者回复: 是的，本质就是这样



👍 1

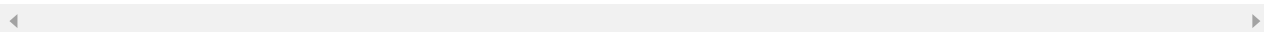


木瓜777

2019-07-13

您好，对model层的理解还是不够，我原先最早认为就是数据结构层，您说把model层做厚的含义是什么？

作者回复: 业务逻辑层，也是整个系统的核心



**1900**

2019-07-12

为啥桌面程序员也叫大前端呢？（另外到底啥叫大前端呢？）

我的初步理解是和用户打交道的软件开发叫前端开发，我的理解对么？

最后，有“大后端”这个术语么？如果有，具体又指什么？

展开 ∨

作者回复: 是的，和用户打交道都是前端或桌面（感觉前端可能更好一些）。叫大前端我理解主要为了强调前端的广度，因为操作系统实在太多了。

**虎哥**

2019-07-12

把多租户的 API 转译成单租户的场景。所以这一层并不需要太多的代码，甚至理论上自动实现也是有可能的。请问哪里有相关资料可以查阅这个内容

展开 ∨

作者回复: 下一讲是一个例子

