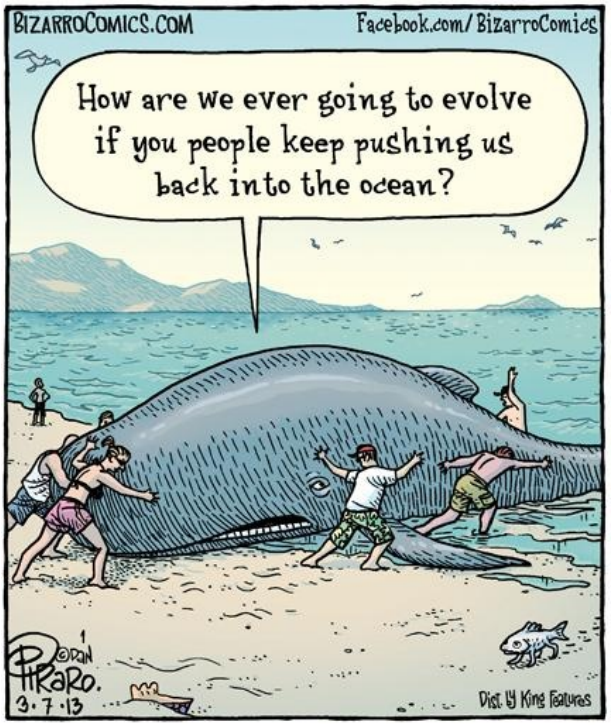
Web 研发模式演变 #184

New issue

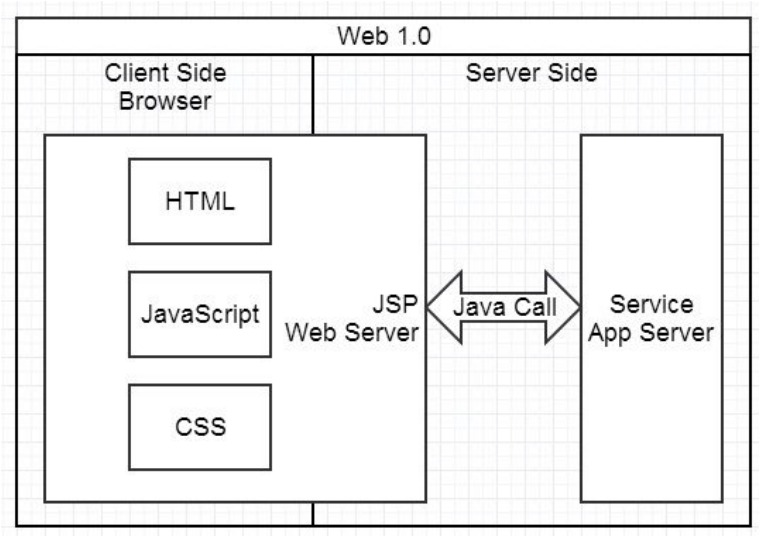
Open **lifesinger** opened this issue on 13 Jan 2014 · 146 comments

lifesinger commented on 13 Jan 2014 Owner



前不久徐飞写了一篇很好的文章：Web 应用的组件化开发。本文尝试从历史发展角度，说说各种研发模式的优劣。

一、简单明快的早期时代



可称之为 Web 1.0 时代，非常适合创业型小项目，不分前后端，经常 3-5 人搞定所有开发。页面由 JSP、PHP 等工程师在服务端生成，浏览器负责展现。基本上是服务端给什么浏览器就展现什么，展现的控制在 Web Server 层。

Assignees

No one assigned

Labels

Projects

None yet

Milestone

No milestone

Notifications

Subscribe

You're not receiving notifications from this thread.

106 participants

and others

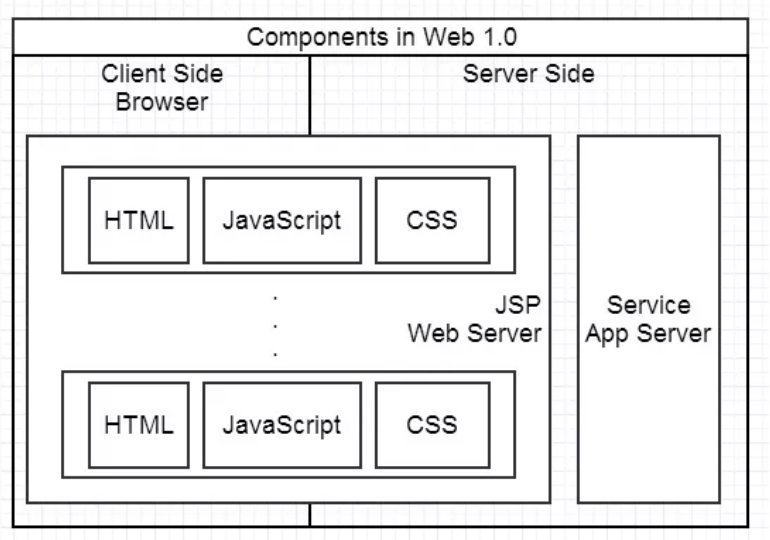
这种模式的好处是：简单明快，本地起一个 Tomcat 或 Apache 就能开发，调试什么的都还好，只要业务不太复杂。

然而业务总会变复杂，这是好事情，否则很可能就意味着创业失败了。业务的复杂会让 Service 越来越多，参与开发的人员也很可能从几个人快速扩招到几十人。在这种情况下，会遇到一些典型问题：

1、**Service 越来越多，调用关系变复杂，前端搭建本地环境不再是一件简单的事。**考虑团队协作，往往会考虑搭建集中式的开发服务器来解决。这种解决方案对编译型的后端开发来说也许还好，但对前端开发来说并不友好。天哪，我只是想调整下按钮样式，却要本地开发、代码上传、验证生效等好几个步骤。也许习惯了也还好，但开发服务器总是不那么稳定，出问题时往往需要依赖后端开发搞定。看似仅仅是前端开发难以本地化，但这对研发效率的影响其实蛮大。

2、**JSP 等代码的可维护性越来越差。**JSP 非常强大，可以内嵌 Java 代码。这种强大使得前后端的职责不清晰，JSP 变成了一个灰色地带。经常为了赶项目，为了各种紧急需求，会在 JSP 里揉杂大量业务代码。积攒到一定阶段时，往往会带来大量维护成本。

这个时期，为了提高可维护性，可以通过下面的方式实现前端的组件化：

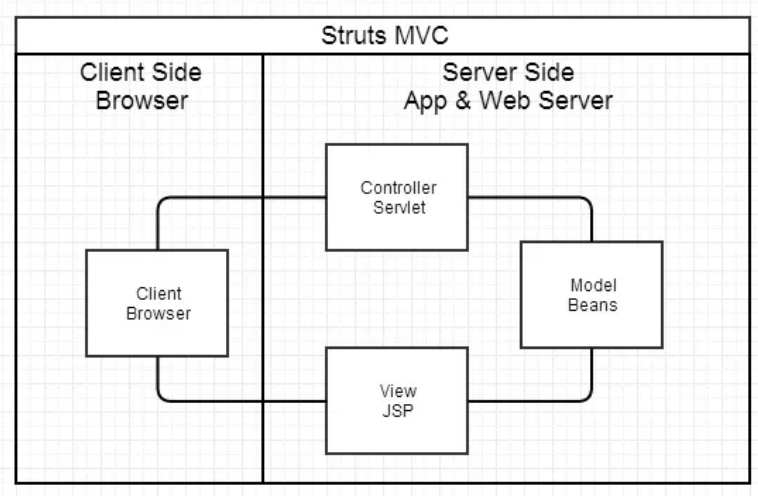


理论上，如果大家都能按照最佳实践去书写代码，那么无论是 JSP 还是 PHP，可维护性都不会差。但可维护性更多是工程含义，有时候需要通过限制带来自由，需要某种约定，使得即便是新手也不会写出太糟糕的代码。

如何让前后端分工更合理高效，如何提高代码的可维护性，在 Web 开发中很重要。下面我们继续来看，技术架构的演变如何解决这两个问题。

二、后端为主的 **MVC** 时代

为了降低复杂度，以后端为出发点，有了 Web Server 层的架构升级，比如 Structs、Spring MVC 等，这是后端的 MVC 时代。



代码可维护性得到明显好转，MVC 是个非常好的协作模式，从架构层面让开发者懂得什么代码应该写在什么地方。为了让 View 层更简单干脆，还可以选择 Velocity、Freemaker 等模板，使得模板里写不了 Java 代码。看起来是功能变弱了，但正是这种限制使得前后端分工更清晰。然而依旧并不是那么清晰，这个阶段的典型问题是：

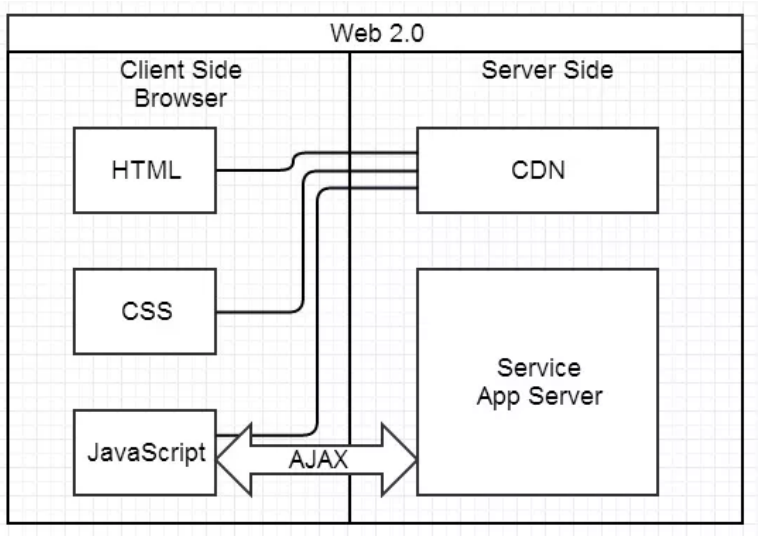
1、**前端开发重度依赖开发环境。**这种架构下，前后端协作有两种模式：一种是前端写 demo，写好后，让后端去套模板。淘宝早期包括现在依旧有大量业务线是这种模式。好处很明显，demo 可以本地开发，很高效。不足是还需要后端套模板，有可能套错，套完后还需要前端确定，来回沟通调整的成本比较大。另一种协作模式是前端负责浏览器端的所有开发和服务器端的 View 层模板开发，支付宝是这种模式。好处是 UI 相关的代码都是前端去写就好，后端不用太关注，不足就是前端开发重度绑定后端环境，环境成为影响前端开发效率的重要因素。

2、**前后端职责依旧纠缠不清。**Velocity 模板还是蛮强大的，变量、逻辑、宏等特性，依旧可以通过拿到的上下文变量来实现各种业务逻辑。这样，只要前端弱势一点，往往就会被后端要求在模板层写出不少业务代码。还有一个很大的灰色地带是 Controller，页面路由等功能本应该是前端最关注的，但却是由后端来实现。Controller 本身与 Model 往往也会纠缠不清，看了让人咬牙的代码经常会出现在 Controller 层。这些问题不能全归结于程序员的素养，否则 JSP 就够了。

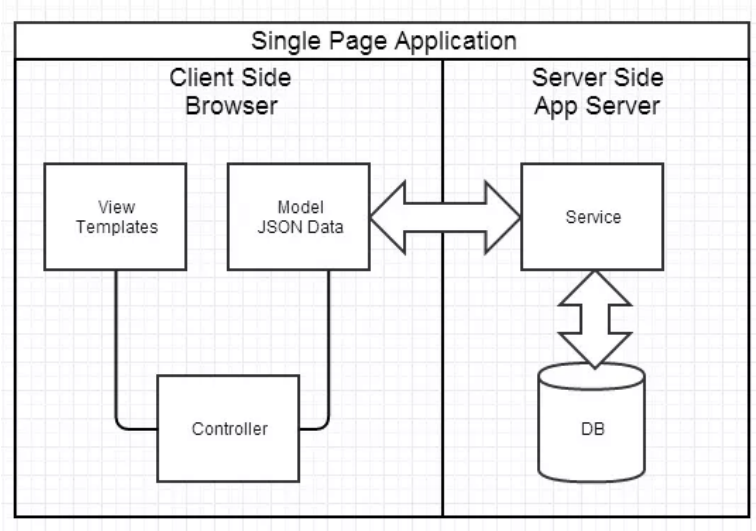
经常会有人吐槽 Java，但 Java 在工程化开发方面真的做了大量思考和架构尝试。Java 蛮符合马云的一句话：让平凡人做非凡事。

三、**Ajax** 带来的 **SPA** 时代

历史滚滚往前，2004 年 Gmail 像风一样的女子来到人间，很快 2005 年 Ajax 正式提出，加上 CDN 开始大量用于静态资源存储，于是出现了 JavaScript 王者归来的 SPA （Single Page Application 单页面应用）时代。



这种模式下，前后端的分工非常清晰，前后端的关键协作点是 Ajax 接口。看起来是如此美妙，但回过头来看看的话，这与 JSP 时代区别不大。复杂度从服务端的 JSP 里移到了浏览器的 JavaScript，浏览器端变得很复杂。类似 Spring MVC，这个时代开始出现浏览器端的分层架构：



对于 SPA 应用，有几个很重要的挑战：

1、**前后端接口的约定。**如果后端的接口一塌糊涂，如果后端的业务模型不够稳定，那么前端开发会很痛苦。这一块在业界有 API Blueprint 等方案来约定和沉淀接口，在阿里，不少团队也有类似尝试，通过接口规则、接口平台等方式来做。有了和后端一起沉淀的接口规则，还可以用来模拟数据，使得前后端可以在约定接口后实现高效并行开发。相信这一块会越做越好。

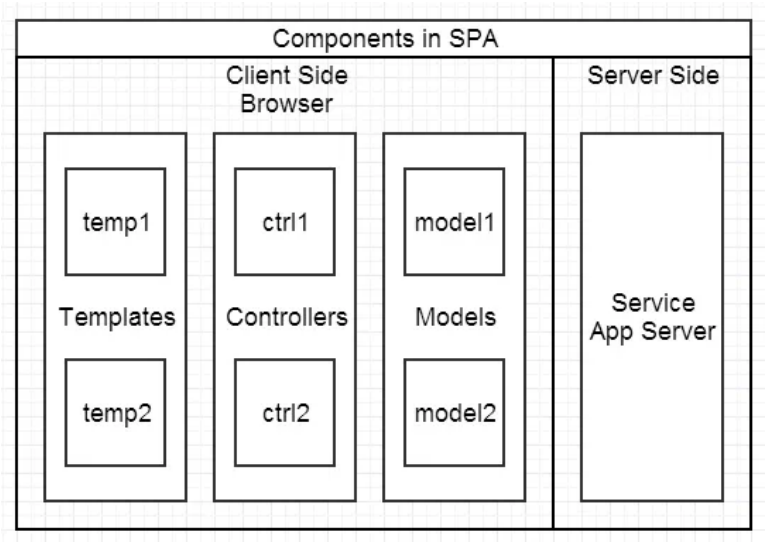
2、**前端开发的复杂度控制。**SPA 应用大多以功能交互型为主，JavaScript 代码过十万行很正常。大量 JS

代码的组织，与 View 层的绑定等，都不是容易的事情。典型的解决方案是业界的 Backbone，但 Backbone 做的事还很有限，依旧存在大量空白区域需要挑战。

SPA 让前端看到了一丝绿色，但依旧是在荒漠中行走。

四、前端为主的 **MV*** 时代

为了降低前端开发复杂度，除了 Backbone，还有大量框架涌现，比如 EmberJS、KnockoutJS、AngularJS 等等。这些框架总的原则是先按类型分层，比如 Templates、Controllers、Models，然后再在层内做切分，如下图：



好处很明显：

1、**前后端职责很清晰。**前端工作在浏览器端，后端工作在服务端。清晰的分工，可以让开发并行，测试数据的模拟不难，前端可以本地开发。后端则可以专注于业务逻辑的处理，输出 RESTful 等接口。

2、**前端开发的复杂度可控。**前端代码很重，但合理的分层，让前端代码能各司其职。这一块蛮有意思的，简单如模板特性的选择，就有很多很多讲究。并非越强大越好，限制什么，留下哪些自由，代码应该如何组织，所有这一切设计，得花一本的厚度去说明。

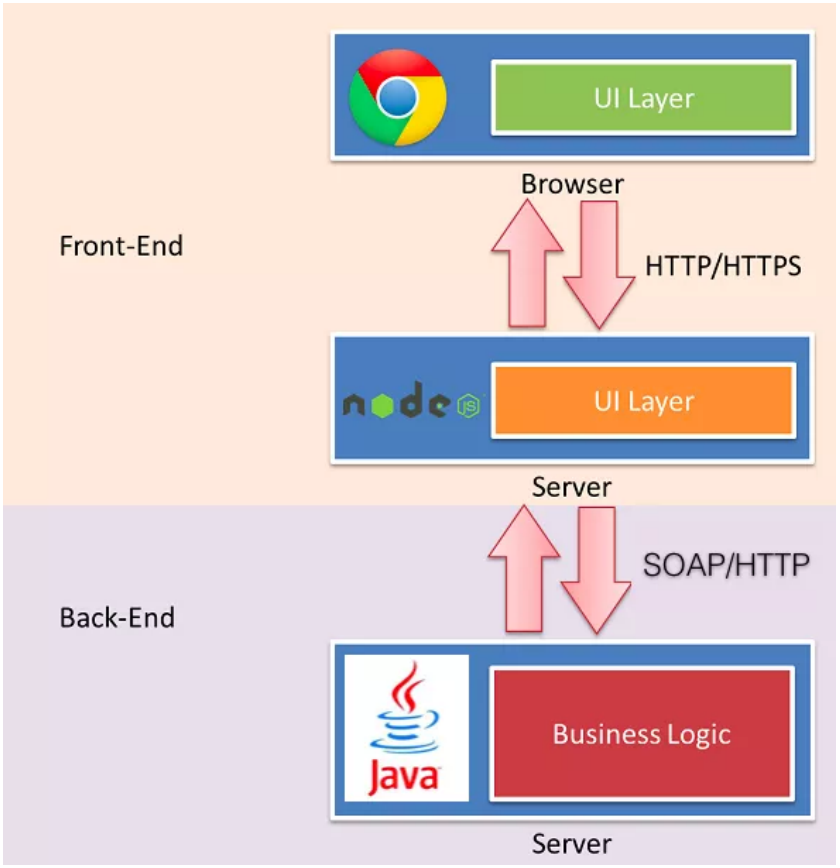
3、部署相对独立，产品体验可以快速改进。

但依旧有不足之处：

1、代码不能复用。比如后端依旧需要对数据做各种校验，校验逻辑无法复用浏览器端的代码。如果可以复用，那么后端的数据校验可以相对简单化。
2、全异步，对 SEO 不利。往往还需要服务端做同步渲染的降级方案。
3、性能并非最佳，特别是移动互联网环境下。
4、SPA 不能满足所有需求，依旧存在大量多页面应用。URL Design 需要后端配合，前端无法完全掌控。

五、**Node** 带来的全栈时代

前端为主的 MV* 模式解决了很多很多问题，但如上所述，依旧存在不少不足之处。随着 Node.js 的兴起，JavaScript 开始有能力运行在服务端。这意味着可以有一种新的研发模式：



在这种研发模式下，前后端的职责很清晰。对前端来说，两个 UI 层各司其职：

1、Front-end UI layer 处理浏览器层的展现逻辑。通过 CSS 渲染样式，通过 JavaScript 添加交互功能，HTML 的生成也可以放在这层，具体看应用场景。

2、Back-end UI layer 处理路由、模板、数据获取、cookie 等。通过路由，前端终于可以自主把控 URL Design，这样无论是单页面应用还是多页面应用，前端都可以自由调控。后端也终于可以摆脱对展现的强关注，转而可以专心于业务逻辑层的开发。

通过 Node，Web Server 层也是 JavaScript 代码，这意味着部分代码可前后复用，需要 SEO 的场景可以在服务端同步渲染，由于异步请求太多导致的性能问题也可以通过服务端来缓解。前一种模式的不足，通过这种模式几乎都能完美解决掉。

与 JSP 模式相比，全栈模式看起来是一种回归，也的确是一种向原始开发模式的回归，不过是一种螺旋上升式的回归。

基于 Node 的全栈模式，依旧面临很多挑战：

1、需要前端对服务端编程有更进一步的认识。比如 network/tcp、PE 等知识的掌握。
2、Node 层与 Java 层的高效通信。Node 模式下，都在服务器端，RESTful HTTP 通信未必高效，通过 SOAP 等方式通信更高效。一切需要在验证中前行。
3、对部署、运维层面的熟练了解，需要更多知识点和实操经验。
4、大量历史遗留问题如何过渡。这可能是最大最大的阻力。

六、小结

回顾历史总是让人感慨，展望未来则让人兴奋。上面讲到的研发模式，除了最后一种还在探索期，其他各种在各大公司都已有大量实践。几点小结：

1、模式没有好坏高下之分，只有合不合适。
2、Ajax 给前端开发带来了一次质的飞跃，Node 很可能是第二次。
3、SoC（关注度分离） 是一条伟大的原则。上面种种模式，都是让前后端的职责更清晰，分工更合理高效。
4、还有个原则，让合适的人做合适的事。比如 Web Server 层的 UI Layer 开发，前端是更合适的人选。

历史有时候会打转，咋一看以为是回去了，实际上是螺旋转了一圈，站在了一个新的起点。

（完）

题图：演化真不容易呀。

欢迎扫描二维码订阅：

👍 50 👎 1 ❤️ 4

viccici commented on 13 Jan 2014

合适的时间与合适的技术。

❤️ 3

alvinhui commented on 13 Jan 2014

最后一个全栈时代，我厂也有一些实践。社会分工粒度越来越细了，对于提高生产力确实有很大的促进作用，互联网企业很快迎来“工厂时代”。但是站在个人的角度去看，又觉得有点儿沮丧.....

两个别扭的：

第四点不属于“Web 研发模式演变”这个主题吧。。。

“2、Ajax 给前端开发带来了一次质的飞跃，Node 很可能是第二次。” 对于Node的论述不同意。

==========我是分割线小朋友==========

可惜看玉伯文章的大多数都是前端程序员。要是架构师、后端们也来关注一下前端的圈子，很多事情才能更快地推动起来。

👍 4 ❤️ 2

guilipan commented on 14 Jan 2014

nodejs在处理复杂的业务逻辑确实不适合,但是处理显示逻辑简直是神器

YSlove commented on 14 Jan 2014

怎么说呢，只有适合跟不适合，善用与不善用。应用为王，其他都是支持为主。没有对错，只有支持的速度与质量差异。任何程序说白了都是利益链条，无利不起早。

huangyingjie commented on 14 Jan 2014

前端操控路由这块，如果前端也掌握java或者php，这不是问题，问题是，前端有没有涉足这块的权力；前后端都用js，未必就能提高可维护性。前后端共用同一代码，这个不错。

👍 2 😄 1

dqw commented on 14 Jan 2014

前端后端的最大区别其实是知识体系不同，和用什么语言关系不大
后端学习javascript困难不大，其实很快就能上手，相对前端学习整个后端的知识体系会困难很多吧
彼此学习一些对方的东西沟通效率会高很多
我是后端，最烦套模板了，前后端两遍验证确实很麻烦，有时候修改还造成不一致

xiongsongsong commented on 14 Jan 2014

@alvinhui Node的确很可能是第二次。

其一：Ajax很不爽的一点是，比同步慢。要客户端发起至少第2次请求后，才能得到结果。如果用Node，可以直接在服务器将数据组装好返回。这一点咋看起来没什么意思，但想象的空间很大，意味着前端可以自主实现一些业务，开发专注于接口的实现。

其二：有必要提下JSON，JSON数据表达能力超强，而不像干瘪瘪的二维表。开发说，写SQL的时候，把数据想象成一个平面。这是没办法的事情，因为二维表就是行和列，数据表现离不了主键外键。JSON就不一样了，它是作为数据交换和表达的极为理想的格式，至少目前我这么觉得。如果条件允许，如果当前存在MongoDB的DBA，那么前端的研发模式的确要转变了。

qqqzhch commented on 14 Jan 2014

好文 收藏了

xiongsongsong commented on 14 Jan 2014

@dqw 前端的知识体系里面，JS只是其中一部分呢。

xufei commented on 14 Jan 2014

最后这个图是月影翻译的那篇文章中的，我也一直在思考最后一步，可能会搬一套类似npm的东西到上面去，但是开发阶段跟运行阶段的分界在哪里，还在纠结。

qqqzhch commented on 14 Jan 2014

看完了 不错

houfeng0923 commented on 14 Jan 2014

前段时间也对web应用开发模式做了一些回顾和总结，分类也大致相当，但远没玉伯老师总结的清晰、条理。学习了。

第五点（Node带来的全栈时代）确实是个让人兴奋的模式，去年yahoo开源了基于yui3的 mojito 应该算是这种模式的尝试。只不过它更彻底，后端完全基于Node.js 。

hanzheng commented on 14 Jan 2014

我厂走到四了，五还不想尝试。。。

elover commented on 14 Jan 2014

本来想走到4但是因为兼容性，认证等问题，现在想试试5，但是目前这种解决方案还太少，到时候做的时候可能也会有不少的坑吧？而且，性能上有可能会比较慢一下，最后前端要介入以前属于后端的一些东西阻力还是比较大的。

shiran565 commented on 14 Jan 2014

对于5的确非常感兴趣，但是很难去说服管理层去做出决策去改变，决策者一般不会愿意仅仅为改变开发模式而付出太大代价，除非它能带来效率上的极大提升

lifesinger commented on 14 Jan 2014 **Owner**

支付宝已经在尝试了，打动管理层有两点：1）性能提升、成本降低，如果能证明通过 Node 可以降低服务器数量，那么成本的节省会比较可观。2）研发效率的提升，前后端都更快速，如果能拿到数据比对，也会非常不错。

2014/1/14 shrian notifications@github.com

> 对于5的确非常感兴趣，但是很难去说服管理层去做出决策去改变，决策者一般不会愿意仅仅为改变开

> 发模式而付出太大代价，除非它能带来效率上的极大提升
>
> —
> Reply to this email directly or view it on GitHubhttps://github.com/lifesinger/lifesinger.github.com/issues/184#issuecomment-32242724
> .

王保平 / 玉伯（射雕）
送人玫瑰手有余香

Wuvist commented on 14 Jan 2014

印象中paypal也走了全栈这条路：http://www.infoq.com/news/2013/11/paypal-java-javascript

做为后端开发者，我对node本身其实很不看好，纯粹考虑后端的话，我会认为后端可以其它有比node更好的选择。就性能提升、成本降低这点而言，node可以做到的，还会有别的方案也可以做到，甚至更好做好。

前后端的代码复用，大头应该是模板，而这层应该可以通过使用mustache之类语言无关的模板引擎达到同样效果。

我会更加期待看到一些后端非node，并且跟前端共享模板的方案；个人两年前也用py在 http://meishiwanjia.com 做过尝试，后端的tornado跟前端的jquery（当然，还用了seajs～）使用同样的模板: http://www.slideshare.net/Wuvist/tornado-9296213 slide 29+

当然，综合而言，肯定还是js + node这样同一语言能够达到的开发效率会更高。但如果前后端都是同一开发者负责开发，从管理的角度看，我猜未必会更好：全栈开发者本身可能会成为瓶颈。

zzxadi commented on 14 Jan 2014

任何技术的发展都有一个类似的过程，想起来《Unix 编程艺术》中开篇的一句话，不懂Unix的人最终还会发明一个蹩脚的Unix。

👍 2

lifesinger commented on 14 Jan 2014 Owner

@Wuvist mustache 等方式能共享的是模板片段，不能从页面整体来优化模板，难以享受 block、filters 等特性。整体化的模板替换方案，对文件组织带来的好处更多，对效率的提升更大。mustache 等方案有点为了共享而共享，牺牲太多。

之所以我觉得是 node，是因为 javascript，因为熟悉，加上适度侵入服务端（不是把服务端的活都干了，而是只负责后端 view、router 等前端相关的活），这种全栈非常谨慎，依旧需要后端的全力配合，核心是让合适人干合适的事，是分工的重新思考和细节调整，因此我觉得是人员上是可控的，可以尝试的。

Wuvist commented on 14 Jan 2014

@lifesinger 重新把你原文看了几遍；应该终于理解你意思了~~哈哈~~不过，我反过来却又会怀疑"性能提升、成本降低，如果能证明通过 Node可以降低服务器数量"这目标。

原来的后端直接渲染jsp模板，性能应该要比由后端输出数据，然后node再渲染模板要好。不管node本身性能如何，它在这里是新的一层overhead。

chemzqm commented on 15 Jan 2014

使用node共享的不仅是模板，我们项目里面有许多模块都是复用的，比如http请求使用superagent，日期处理使用momentjs，异步处理使用serial和parallel（一开始使用的promise也是完全一样的代码），另外还有JSON解析的一致性也是其它语言做不到的。

xufei commented on 15 Jan 2014

@Wuvist 性能未必好，重点是把可能占应用服务器资源的一些非关键代码抽到外面来，比如模板、国际化这些东西，应用服务器应专注于处理真正的业务，其性能不应被这些外部的东西干扰

Wuvist commented on 15 Jan 2014

@xufei 但按 **@lifesinger** 的说法，“性能提升、成本降低，如果能证明通过 Node 可以降低服务器数量，那么成本的节省会比较可观”是“打动管理层”支持尝试加入node做全栈的两点理由之一。

不过换个角度看，很多公司（比方我现在的公司）应该也是ajax前端一层，然后php UI一层，然后再是后端各种service。其中ajax前端由前端团队负责；php + 各种service由后端团队负责。那么，玉伯说的5，相对于这样的架构，是把php换成了node，并且负责的团队由后端变成前端。

node相对于php，也许可以获得性能提升；并且还有各种代码共享的好处。但相应的问题是如 **@chemzqm** 所说的node人员不好招。

我一直都对node不感冒，因为我实在不觉得它是一个后端开发的好选择；但看到这篇文后，对于 **@lifesinger** 提出的这种把node做为UI渲染的“后端服务器"的使用方式，我觉得还是很有道理，很值得尝试的。

以后端开发而言，我会更加看好go，但目前go在UI渲染方面还是比较弱，把go挪去“真·后端”，UI渲染换成node，我觉得会是挺不错的选择。

lifesinger commented on 15 Jan 2014 Owner

@Wuvist 可以无视我的理由，之所以提那两点，是因为这两点是短期内有可能看得见的。引入 node 的全栈模式，更多受益在长期，比如

1. 前端的研发效率，这个长期才能看得出来，起步阶段需要的是相信。
2. 逼着前后端协作往接口化靠，甚至把接口都 RESTful 化。这个光口头叫叫是做不出来的，得研发模式中有破局之处。

招聘的问题，直接招很难，内部培养还是相对容易的。需要的是团队里有1-2个专家级人物，然后逐步培养，逐步吸引。

Wuvist commented on 15 Jan 2014

@lifesinger "逼着前后端协作往接口化靠，甚至把接口都 RESTful 化。这个光口头叫叫是做不出来的，得研发模式中有破局之处。"咦？我以为阿里/淘宝/支付宝早就SOA了？

lifesinger commented on 15 Jan 2014 Owner

UI 层没接口化，还是 Velocity 变量搅拌成一团

2014/1/15 Wuvist notifications@github.com

> **@lifesinger** https://github.com/lifesinger "逼着前后端协作往接口化靠，甚至把接口都 RESTful 化。这个光口头叫叫是做不出来的，得研发模式中有破局之处。"咦？我以为阿里/淘宝早就SOA了？
>
> —
> Reply to this email directly or view it on GitHubhttps://github.com/lifesinger/lifesinger.github.com/issues/184#issuecomment-32330628
> .

王保平 / 玉伯（射雕）
送人玫瑰手有余香

👍 1

chemzqm commented on 15 Jan 2014

从付出、收益来讲小公司根本培养不起，因为受众太少，而且还容易跑路，让专业的人付出太多精力去做培训得不偿失。往全端走障碍还是蛮多的，首先是前端上光会jquery underscore那些工具肯定远远不够，然后是前后端API设计上的得失考量，学会node里面那些常用模块和API我个人觉得是最容易的，建议基础不是太好的同学先把语言和协议的基础打好一些，不要想着一步跨入全端。

On 2014年1月15日, at 10:50, lifesinger notifications@github.com wrote:

> **@Wuvist** 可以无视我的理由，之所以提那两点，是因为这两点是短期内有可能看得见的。引入 node 的全栈模式，更多受益在长期，比如
>
> 前端的研发效率，这个长期才能看得出来，起步阶段需要的是相信。
> 逼着前后端协作往接口化靠，甚至把接口都 RESTful 化。这个光口头叫叫是做不出来的，得研发模式中有破局之处。
> 招聘的问题，直接招很难，内部培养还是相对容易的。需要的是团队里有1-2个专家级人物，然后逐步培养，逐步吸引。
>
> —
> Reply to this email directly or view it on GitHub.

xingrz commented on 16 Jan 2014

（虽然之前在微博提问过，但感觉这里问的话讨论氛围好一些。）

我们团队也是采用第五种开发模式，后端使用 Golang。

总体上大家都用得很愉快，但是我有一个地方比较纠结：用户会话应该由后端还是 Node 来维护？

如果由后端维护

后端生成会话 token 丢给 Node；Node 将 token 存入 Cookie，每次请求时发送给后端作为用户身份鉴别，以此来获取用户 id。

此时 Node 的职责仅仅是渲染模板、处理路由，纯粹是浏览器也业务后端之间的一个中转器。

如果由 **Node** 维护

由 Node 生成会话 token（此时的 token 就是大家理解的 Session 存到 Cookie 里的 Session ID 了），由 Node 维护用户 id 等会话数据（可能使用 redis 做持久化）；Node 与后端的通讯直接使用用户 id 来区分用户，也就是说后端无条件信任 Node 带过来的用户 id。

此时对于 Node 来说，可以把后端看做一个「数据库」。后端只保证进出的数据在业务上的正确性，而不处理会话。

不知道各位对这两种方式有什么看法？

lifesinger commented on 16 Jan 2014 Owner

@xingrz 支付宝目前想尝试第一种方式：session 由后端维护，Node 通过 node-tair 模块去读取 session 信息。这样做是基于支付宝目前的技术体系，后端 session 层已经很复杂，比如分布式 session 等，这些逻辑如果用 Node 重新实现，代价很大。

如果是轻量级应用，session 由 Node 来维护可能也是不错的选择，这个我们没尝试，就不评价了。

xingrz commented on 16 Jan 2014

谢谢玉伯老师的回复。

实际上我们团队现在用到这种结构的两个产品都是由后端维护 session。

当时原本的结构是后端完全用 go 实现，后来发现 go 渲染 html 实在痛苦，才加了一层 node；而且当时还打算手机客户端直接与 go
后端打交道，所以就把用户会话做在 go 了。

直到后来再仔细思考这种结构的时候才想到了由 node 维护 session 的方式，因为这样对于「明显不合法」的请求，比如无效 session
id，可以直接在 node 层拒掉，少走两段路程。而且后端也不应该暴露给外网，诸如手机客户端用的接口需要再包一层认证和会话维护。

另外当时对这种方式的想法还有，后端与 node 可以使用诸如 protobuf 之类更高效的序列化协议，协议带上版本等。不过这些还停留在设想。

在地铁上码字，可能有点不通顺。
2014年1月16日 上午9:23于 "lifesinger" notifications@github.com写道：

@xingrz https://github.com/xingrz 支付宝目前想尝试第一种方式：session 由后端维护，Node 通过 node-tair 模块去读取 session 信息。这样做是基于支付宝目前的技术体系，后端 session 层已经很复杂，比如分布式 session
等，这些逻辑如果用 Node 重新实现，代价很大。

如果是轻量级应用，session 由 Node 来维护可能也是不错的选择，这个我们没尝试，就不评价了。

—
Reply to this email directly or view it on GitHubhttps://github.com/lifesinger/lifesinger.github.com/issues/184#issuecomment-32434110
.

Wuvist commented on 16 Jan 2014

@xingrz js/go的话，thrift的支持要比pb更好。最近刚好试用了pb跟thrift，pb无map数据结构支持，无rpc定义支持，多语言支持较弱等这三点让偶选择了thrift。

性能方面thrift也完全不比pb差；它有binary compact/binary/json三种序列化方式，其中binary compact的性能绝不会逊于pb；而json则让js方便得多。

xingrz commented on 16 Jan 2014

谢谢 **@Wuvist** ，记下了。我现在也只是设想阶段，还没具体比较过。有机会会试试。

nuintun commented on 16 Jan 2014

记得QQ空间V8就是这样做的！

dYb commented on 4 Mar 2014

我现在的做法是，开发时，用node调用freemarker和velocity的jar包，前端直接写对应的ftl和vm模板。这就需要前后端的先约定好借口，可以节省很多时间。

jieqiuming commented on 15 Apr 2014

看完全文以及这么多大牛的评论，真的长知识了，感谢!

magicdawn commented on 22 Apr 2014

js感觉写起来不好组织,一大坨代码堆一起,蛋疼...java/C# 是一个package含多个类,到node这一个类就可以弄个module,还有npm太开放了...一堆没用的,估计

joostshao commented on 23 Apr 2014

callback in callback,
you his morther, is kidding me?

luck2011 commented on 23 Apr 2014

长知识

chenkan commented on 26 Apr 2014

文章写得很赞，楼主看得很通透
一点小瑕疵，如下所示，加粗格式没处理好，全文一共有三处

2、**JSP 等代码的可维护性越来越差。**JS
不清晰，JSP 变成了一个灰色地带，经常为

lostlic commented on 29 Apr 2014

这种模式实际上就是把后端的portal应用用node来写嘛，对于纯粹的前端+后端的组合确实效率提升不少。但对于既会前端又会后台的意义不大。

hoosin commented on 29 Apr 2014

@chenkan 哈哈哈，吹毛求疵

artman328 commented on 3 May 2014

如果开发的是业务管理应用，Vaadin 的优势是非常明显的！

magicly commented on 3 May 2014

我觉得SPA阶段里面那张前端MVC的图片应该已经属于文中4阶段了吧。

最近在做一个cms，痛苦的来回在前后端之间切换，然后修改一个颜色什么的，需要同步代码，发布，查看复杂的步骤，一遍要好几分钟。

第四阶段的分层方法我觉得对于cms来说very good了，性能不是问题，SEO也不需要。前端打算用AngularJS，后端用Spring REST，看看效果哈~

FrankyYang commented on 4 May 2014

全栈模式需要后端先有所进化，比如有一个相对统一的数据仓库，通用的基于RPC或SOAP的存取等。然后为数据仓库提供一个相对靠前的编辑系统，数据关系处理的逻辑在这个系统中整理清楚。最后就可以向用户端开放，由全栈式的系统主导的模块。

xufei commented on 4 May 2014

@magicly 是这样，如果你的所有东西都能走AJAX之类的通道，后端是完全的接口化，而且也不要SEO，确实第四层是非常够的。第5层面对的场景是前后端模板同时存在之类，架构的复杂度更高了

RobinQu commented on 6 May 2014

说实话，看到这篇文章感觉很失望。LZ提出的每种架构，都早已不是什么新东西。

我大概理解LZ在最后提出的Java应用作为SOA来支撑Node的"前端"是什么个意思：说白了就是企业架构已经被Java技术栈所占领，node开发（或者Go、Python等非Java语言）只能适应别人。其结果：

1. 忍耐缓慢的http接口去提供前端服务
2. 大量的服务器资源浪费
3. 实际整体架构变复杂了

我目前看到比较新的的技术或者理念：

1. Database as interface，典型例子：CouchDB，各种view直接将数据输出为json、html等
2. WebComponent，新的组件封装形式，虽然是前端技术，但是服务器可以利用这个封装体系统一输出样式、行为、数据，有点GWT的味道，但是比GWT更强大、开放
3. .....

不考虑这些太潮的东西，异构系统利用thrift或者其他统一协议汇总到总线，再提供http或者websocket这种用户协议的架构，或许是更精简、现实的方案。

我这抛砖引石了，其实web开发很久都没有什么新东西了，都是"微创新"。

darkjune commented on 9 May 2014

总结的不错

alongytl commented on 22 May 2014

“全栈模式"这个词谁想出来的？
nodejs这一层，咋看就是个适配器，不过也增加了复杂度，会不会出现后台服务变化之后，nodejs和前端都得一起调整？
效果到底咋样，等实践结果。我是觉得架构要越简单越好，RobinQu 的建议比较赞同

aufula commented on 15 Jun 2014

看过一个conf的视频, 是关于papal的前端jsp到node的改建。一个不错的例子。
最后特别强调的是需要招大批强大的前端，doug也被拉去做培训师了。
如果用node的模式来开发，工作量，学习的范围和成本都会成倍增长，还得考虑把业内顶尖的前(/全)端都招来，才能付诸实行，hoho。

brainee commented on 27 Jun 2014

我基本上算全的了吧，前端后端都涉猎，前端主要是范，散，不规范，后端用node做中间件的确不错，一来可以继承历史资产，用node和c#，java通过协议交互或者直接调用dll交互，二来node的无阻塞高并发事件驱动非常有优势，第三node与前端配合能实现更好的业务搭配，如某些资源单纯的用Ajax去调用，远不如node在后端生成好，直接抛给前台速度快，而且在极端网络环境下更明显，这也有写.net的思想在里面。

freestyle21 commented on 27 Jun 2014

@lifesinger

1. 请问一下图是用什么画的。
2. markdown的强调『**』有两处没其作用，看着挺别扭的。

xufei commented on 3 Jul 2014

@freestyle21 方块图是用chrome浏览器商店的Gliffy Diagrams画的

norfish commented on 8 Jul 2014

我个人的理解：其实这么多模式的转变无非是围绕着两个核心问题：UI层该由谁负责渲染、UI交互应该谁来处理。特别是UI层的渲染问题，往往会造成前后端划分不明确的问题，JSP和ajax混杂。所以才有了第五种模式，由前端全部接管UI层的处理。

这阵子在看facebook的react方案，如果用react在服务端去处理UI层的问题的话，会是一个职责分离的例子。

aufula commented on 8 Jul 2014

Facebook的开发人员基本都是全端
2014年7月8日 PM2:33于 "Yesol.Lee" notifications@github.com写道：

> 我个人的理解：其实这么多模式的转变无非是围绕着两个核心问题：UI层该由谁负责渲染、UI交互应该谁来处理。特别是UI层的渲染问题，往往会造成前后端划分不明确的问题，JSP和ajax混杂。所以才有了第五种模式，由前端全部接管UI层的处理。
>
> 这阵子在看facebook的react http://facebook.github.io/react/
> 方案，如果用react在服务端去处理UI层的问题的话，会是一个职责分离的例子。
>
> —
> Reply to this email directly or view it on GitHub
> lifesinger#184 (comment)
> .

hanzheng commented on 8 Jul 2014

any comments on this: http://www.zhihu.com/question/23512853

StevenX911 commented on 9 Jul 2014

我实在看不出第四和第五种模式之间的本质差别在哪？？难道仅仅这是语言的不同吗？使用Node可以实现的所有功能，无论 Stucts 或 Spring MVC 抑或是 Asp.net MVC等等模式 同样完全可以做到！拉取你想要的视图，定义你想要的路由，实现你想要的SEO，无论是单页面或者多页面，这些都不是问题！另外在性能上或者开发效率都完胜Node.js(V8). 解决方案也相对更多更成熟！

不过，使用Node.js 确实可以缩短战线，前端可以只保留UED和JS，而后端只关注数据接口，减除了JSP等中间层。这确实可以是个好处，但许多问题都有待解决。

datianyun referenced this issue in **datianyun/datianyun.github.io** on 11 Jul 2014

前端集成解决方案 #4 Open

ghost commented on 15 Jul 2014

把移动互联网置于何地，难道不算web？

shiningguang commented on 19 Aug 2014

mark

willin commented on 1 Sep 2014

mark

xiaolong3000 commented on 1 Sep 2014

mark

passionguy commented on 3 Sep 2014

mark

RobinQu commented on 3 Sep 2014

我也是醉了，一群mark党。LZ能抽空close么？

hoozi commented on 10 Sep 2014

可以转载吗？

shiningguang commented on 11 Sep 2014

真心木有时间看

2014-09-10 16:23 GMT+08:00 Joker notifications@github.com:

> 可以转载吗？
>
> —
> Reply to this email directly or view it on GitHub
> lifesinger#184 (comment)
> .

hanzheng commented on 11 Sep 2014

这种前端架构 加上后端微服务 是一个不错的组合
实践中

shiningguang commented on 11 Sep 2014

嗯，有时间再仔细看，并学习实现

2014-09-11 10:12 GMT+08:00 zheng han notifications@github.com:

> 这种前端架构 加上后端微服务 是一个不错的组合
> 实践中
>
> —
> Reply to this email directly or view it on GitHub
> lifesinger#184 (comment)
> .

yubiaobin commented on 11 Sep 2014

前端的要求 越来越高，要学习的技能 也是越来越多了。

binuy commented on 12 Sep 2014

还是小白，在前端的路上裸奔着，不过现在的感觉就是，要学的东西好多。最近校招让我压力山大啊

在 2014-09-11 11:10:24，"Max yu" notifications@github.com 写道：

前端的要求 越来越高，要学习的技能 也是越来越多了。

—
Reply to this email directly or view it on GitHub.

leson commented on 14 Sep 2014

还处在第二与第三阶段的来取经来了。果然受益匪浅！

luobotang commented on 27 Sep 2014

学习了！

crossjs commented on 29 Sep 2014

楼上自己不努力，却在怪运气

zzz6519003 commented on 29 Sep 2014

@DemonCloud your blog is very cool~~~~

crossjs commented on 29 Sep 2014

@DemonCloud
可以考虑寻找更适合自己、能够让自己更有成长的团队了。
比如 BAT 这些，都有很好的基于业务的经验积累，国内能让你学到东西的团队还是很多的。

DemonCloud commented on 30 Sep 2014

@zzz6519003
Thanks...

ZhibingXie commented on 3 Oct 2014

好文 学习了

tobyreynold commented on 3 Oct 2014

讲的很详细。。。

aherain commented on 13 Oct 2014

说的很好，对于进一步学习web前端，有很好的指导意义

hacke2 referenced this issue in **hacke2/hacke2.github.io** on 22 Oct 2014

HTML5和JavaScript Web应用开发读书笔记 #5 Open

nilyang commented on 23 Oct 2014

相比而言，我更倾向于thrift这种中间件 接口方式协调各端。不论前端（相对服务端，企业服务而言）是用node还是PHP还是Java，或者是C++。

zerosoul commented on 27 Oct 2014

我厂处于三和四的混合模式，个人正在探索第五种模式...

magicwind commented on 28 Oct 2014

可以把第五种模式看成微服务架构的变种，node的UI层本身可以封装成一个微服务：主要负责MVC中的V和C，其次是请求转发。而Java的后台可以演变成多个微服务：分成用户，权限，业务等多个功能模块。

这样node的微服务只调用java的微服务，层次结构还是比较清晰的。

yangluotong commented on 28 Oct 2014

Leave a comment.

This was referenced on 14 Nov 2014

2014，你好11月。 zhangmengxue/Todo-List#4 Open

2014年11月【下】-前端开发半月刊 jikeytang/jikeytang.github.io#12 Open

FrankFan commented on 14 Jan 2015

看完了，总结的很好，多谢分享

winfan commented on 14 Jan 2015

看完了，总结的很好，多谢分享

ZhouYK commented on 16 Jan 2015

点个大赞

hipop commented on 23 Jan 2015

不完全赞同“不是新东西”的观点。对于前端人员，5是入侵到服务器的，具有里程碑意义。
但最大的问题也在这里，也就是文中提到的第一个挑战！5模式对前端开发者提出了更高的要求，工作中，人的问题才是最大的问题。

hdpter commented on 30 Jan 2015

学习了，3Q有大神们

mintisan commented on 1 Feb 2015

喜欢螺旋式前进这种方式～

JianyuC commented on 7 Feb 2015

让我对前端分层更清晰，感谢您的分享

michaeljunlove commented on 14 Feb 2015

mark

JsonSong89 commented on 6 Mar 2015

就架构而言，没感觉5和4 的差距有多大。
区别仅仅是中间层的逻辑由谁来写。。。

panhainan commented on 6 Apr 2015

mark

panhainan commented on 6 Apr 2015

1、模式没有好坏高下之分，只有合不合适。
2、Ajax 给前端开发带来了一次质的飞跃，Node 很可能是第二次。
3、SoC（关注度分离） 是一条伟大的原则。上面种种模式，都是让前后端的职责更清晰，分工更合理高效。
4、还有个原则，让合适的人做合适的事。比如 Web Server 层的 UI Layer 开发，前端是更合适的人选。

历史有时候会打转，咋一看以为是回去了，实际上是螺旋转了一圈，站在了一个新的起点。

恩恩

donkeyjoey commented on 14 Apr 2015

对全栈工程师的话题比较感兴趣，我们公司没有专职的前端工程师，每个工程师都是前后都写，这倒是省了引入node.js这样的中间层的必要，但依然要不断校正UI与业务逻辑分离的状况。

DavidCai1993 commented on 23 Apr 2015

受教

abruzzihraig commented on 27 Apr 2015

「五、Node 带来的全栈时代」中的图文有点问题吧，Back-end UI layer应该是指Node这一层，图里没标注有点歧义。

HelloHack commented on 28 Apr 2015

关于"5 NODE 全栈"
1、Front-end UI layer 处理浏览器层的展现逻辑。通过 CSS 渲染样式，通过 JavaScript 添加交互功能，HTML 的生成也可以放在这层，具体看应用场景。

2、Back-end UI layer 处理路由、模板、数据获取、cookie 等。通过路由，前端终于可以自主把控 URL Design，这样无论是单页面应用还是多页面应用，前端都可以自由调控。后端也终于可以摆脱对展现的强关注，转而可以专心于业务逻辑层的开发。

1和2为什么不用AngularJS实现？比如利用AngularJS数据双向绑定，多视图，路有等策略就来解决这种问题

evanhunt commented on 28 Apr 2015

还停留在第三个阶段准备迈向第四个阶段;
SPA看是很美好,但是感觉有很多局限性,希望能迈向第四阶段
全栈对技术要求何其高,不过对于前端要求越来越高也是一种让人很兴奋的事情.

andot commented on 13 Jun 2015

六、微服务时代
以 hprose 为标志的微服务时代，这个时候前端和后端不但可以很好的分离，后端的代码可以更好的同时复用于后端和前端，前端也从单纯的网页变得更为多样化，同一个后端可以方便的服务于手机app，桌面app，微信app，嵌入式app，网站，甚至是服务器本身。

JsonSong89 commented on 15 Jun 2015

楼上说的就是soa吧.

wxcsdb88 commented on 18 Jun 2015

Great pages!

abnerlinan commented on 19 Jun 2015

看多了总觉得是nodejs的变相的炒作.

amowu commented on 24 Jun 2015

👍

DragWeb commented on 23 Jul 2015

我们公司目录用nodejs+java模式，nodejs这块用了百度的yog2框架，用起来相当舒服

This was referenced on 24 Jul 2015

Web应用的组件化（一）——基本思路 xufei/blog#6 Open

这样算是**blog?** xxg3053/notes#2 Open

sandwich99 commented on 1 Aug 2015

合理利用 IDE 或许是个不错的补充。 将重复、繁琐的沟通 调试工作，交给RAML 这样的 DSL， 使用typescript这样的强类型 javascript， 将代码校验， 接口检查等工作交给IDE

JieChang commented on 2 Aug 2015

学习了

jumplee commented on 23 Aug 2015

通往5的路上..

majianxiong commented on 27 Aug 2015

单页面应用前后端分离，会不会造成前端的工作量变大？

xufei commented on 28 Aug 2015

@majianxiong 虽然前端的工作量大了，但是后端的少了啊。其实这个也看你把怎样的人算成前端，比如说以职责来分，而不是以编写的代码在整体架构中所处位置来分的话，其实并没有变大，因为数据和模板的结合，还有页面的跳转之类，只是从服务端转到了浏览器端而已，带来的好处可能是整个系统的体验更好。

wangchenxiao commented on 9 Sep 2015

C程序员刚转行做web开发，完全看不懂你们在说什么。

SyuTingSong commented on 15 Sep 2015

楼上这条是家里的猫回复的

gaoshunsheng commented on 1 Oct 2015

1、比较同意作者的观点，和我在公司想推行的方案比较类似
2、未来的部门设置可能会把android、ios和web前端以及node（后端）或者java（做数据适配的后端）组成一个大前端部门，这样部门间的沟通会容易很多
3、推荐使用node主要原因还是在于前端学node的成本相对会比较低，当然其实用java、php也可以，只不过用node效率（性能和开发效率）都会高
4、推荐先用熟悉的语言拆分服务层和UI层，然后慢慢让前端对UI层进行重构（使用node），最终达成前后端分离的目的

britrock commented on 8 Oct 2015

当前是一种顺向而势的技术“变革”，按其发展的过程和进程，谈不上有多大的创新。
NodeJS横空出世，让人着实看到了技术开发职能上的转变，拥抱变化，享受HTML+CSS+JS带给我们便利与实际的好处。
在学习的路上.....

ourai commented on 8 Oct 2015

@britrock H5 -> HTML

Jason-mor commented on 4 Nov 2015

目前项目处在楼主说的第四个阶段

feng88724 commented on 12 Nov 2015

好文

Giten2004 commented on 13 Nov 2015

好文，收藏分享

ghost commented on 13 Nov 2015

路过，学习

LilyBlooper commented on 18 Nov 2015

非常好的文章，有醍醐灌顶之效！

tyosssss commented on 23 Nov 2015

刚入门前端，看了这篇文章，对前端的开发模式有了一个大概的了解.
谢谢

dingyiming commented on 23 Nov 2015

我还是有不清楚的地方，请问

- 让nodejs单独作为页面模板渲染来用，但具体是怎么用？
- 怎么和前端MVVM的框架结合用，如何进行共享路由？
- 最好有个具体的代码实例
- 我现在在用vuejs和vue-router，希望做个能解决SEO问题的SPA应用，但不知道有什么现成的代码可以借鉴？angularjs也可以
- 感谢感谢

hoozi commented on 23 Nov 2015

@dingyiming nodejs作为中间层来控制ui层,这样的好处就是前端对路由可控，而且也解决seo的问题，当然这样做前端的工作量也提高了不少

dingyiming commented on 23 Nov 2015

@hoozi 恩恩，它的好处我能理解，但是不知道如何具体实践？特别是希望用上MVVM框架以及组件化开发的情况下，具体实现，不清楚，谢谢。

dingyiming commented on 23 Nov 2015

@hoozi 经过和一些小伙伴讨论，我好像明白了，是我以为最后面的实践方案也是要做成SPA的，其实只不过是把往常的做法分工改了改而已，看来接下来的问题还是得去找找SPA的SEO方案了，谢谢

buyzhi commented on 31 Dec 2015

目前我厂 正在建立模式 菜鸟路过 瞬间醍醐灌顶

djockey commented on 16 Feb 2016

很棒！不愧是专家

This was referenced on 10 Apr 2016

前端入门资源 poetries/mywiki#8 Open

nodejs学习之旅**(**一**)** xxholly32/Blog#3 Open

zerolinke commented on 2 May 2016

两年了.....

heiliuer commented on 4 May 2016

我也非常看好最后的那种方案，希望淘宝能够继续带头探索实践用在生产上。

This was referenced on 14 May 2016

［思考］前端工程化方面的计划 songhlc/blog#1 Open

闲扯前端 yleo77/Notes#6 Open

hujianxin commented on 25 May 2016

非常到位，受益匪浅，谢谢！

calidion commented on 25 May 2016

文章存在这么严重的问题，竟然没有人指出来？
难道现在没有MVC，没有MVVM了？
全栈与MVC还有MVVM是冲突的？
如果不是冲突的何来时代之说？
楼主确定前两个时期的所有服务器都是基于JAVA写的？

👎 2

mlyknown commented on 28 May 2016

@calidion 这篇文章的题目是"Web 研发模式的演变"，包含了服务端和客户端。全栈与MVC,MVVM当然不冲突，服务端和前端，还要node中间层完全可以使用mvc或者mvvm模式。。 这两点并不冲突啊。。。

calidion commented on 28 May 2016 • edited

@mlyknown

首先是什么是研发模式？
研发模式是技术历史还是人的协作模式？

如果是产品开发与项目管理模式来说，也应该是从瀑布到敏捷。
如果是技术演进历史，应该是:
后端:

1. CGI
2. 脚本语言
3. 容器 + 脚本
4. 新型IO模型期

前端：

1. DOM/BOM简单处理时期
2. 富RIA（applet, flash）与无联网js时期
3. AJAX 可实时更新时期
4. HTML5时期(SPA, MVVM等)

我搞不清研发模式是什么，从我的理解看来研发模式更多的应该是技术管理的模式。

其次，MVC、MVVM跟全栈，那个是只有某一时期才有的概念？又是怎么跟研发模式关联上的？

本文，即分不清前后端，也分不清前后端语言。
该抽象不抽象，不该抽象的胡乱抽象。
本质是没有概念，没有逻辑。

MV*无非是MVC的一个迭代模式。
有兴趣也可以参考我对MVC的递归性的阐述。
而全栈一直都有，特别是早期的Web就是全栈的开发模式。

同时全栈更多的是针对人来讲的，而不是技术。
没有nodejs一样可以全栈。
node只是将这种能力要求降低了，并且效率更高了。

还有你所谓的node是中间层就是一个明显的错误概念下的错误理解。
按你们的理解，任何后端语言都可以成为你所谓的node中间层。
所以你提node中间层没有任何意义。如果你将模板解析当成是node中间层，显然你被误导了。
这是一个明显的概念错误。

Web 1.0, Web 2.0完全是营销术语，用于技术上就是牛头对马嘴，狗屁不通。

从文章的内容来看，无非是想表达前后端在Web应用上的比重的变化，以及前后端交互模式的变化，
跟研发模式毛关系没有。

当然更重要的是，作者根本没有清晰的概念，也缺少抽象能力，才能写出来这样的文章。
对于我来讲最重要的，这样的文章误人子弟，害人非浅。
因为被害的人已经不是一个两个了。

在中国概念不清，逻辑不灵是通病。
更要命的是，这些错误还经常是由一些所谓的大厂高手弄出来的。

👍 2 👎 5

calidion commented on 28 May 2016 +😀

通常我能接受基本正确的基础上存在细微的错误，但是无法接受，整体上的逻辑混乱与概念不清。

calidion commented on 28 May 2016 • edited +😀

我在这边列几个比较恶心的术语。
DIV+CSS、大前端、前后端分离。
DIV+CSS是完全错误的概念流行了十几年了，正确的说法应该是HTML+CSS。
估计学前端的90%都分不清楚，这得感谢这么多概念不清的传播者。

大前端，前后端分离是新造的词，无非是搞不清楚前后端是什么，前后端的协作关系是什么，还有MVC与前后端的关系，也有很多是分不清人的职权与技术划分的关系造成的。
大前端根本无法界定。
前后端天然分离，提个毛的前后端分离，无非MVC与前后端分不清楚了，才将MVC的模块分离当成是前后端分离。

👎 9

zproo commented on 3 Jun • edited +😀

看似简单的道理却很少有人能讲的如此清晰。先给徐大佬点赞！
但是加入Node做服务端渲染势必增加了开发复杂度，还有对性能的影响？对稳定性的影响？
阿里已经尝试几年了，现在的迁移过程是什么进度？迁移后效果如何？这个架构是否值得广泛使用？
希望博主以后有时间可以更新一集！关注！！

👎 1

calidion commented on 3 Jun +😀

@zproo

这里认为Node是前端，你问Node服务器端渲染。果然是高人。哈哈。

另：

如果阿里的技术人员有足够的判断，这种错误思想肯定一开始就被灭了。

所以不存在迁移的问题。

并且结果也只有两种可能，一种是后端转向Node，另一种是后端继续使用其它语言。

并不会发生大前端和这里所谓的"前后端分离"这样的事情。

时间确实是最好的证明。 哈哈。

👎 2

yiakwy commented on 4 Jun

我关注这个帖子3年了，等得就是您今天的回答

在 2017年6月3日 下午10:10，Eric <notifications@github.com>写道：

...

👍 1 👎 3

yiakwy commented on 4 Jun

我认为题主自始至终犯了一个错误，就是认为业务人员，就应该用大一统的语言；然后仅仅从技术上看，前段（用户端），后端（高性能，安全）看，采用合适的技术去解决合适的问题，才是计算机从业人员应当思考的事情

在 2017年6月4日 上午1:45，Yiak Wang Yi <yiak.wy@gmail.com>写道：

...

👎 3

yiakwy commented on 4 Jun

很多人把前端的框架画组件技术奉为圭臬，首先我们搞清楚几个问题：

这是不是UI问题？UI讲究的是视觉和渲染效率：
UI的核心技术，你是不是该会用CSS，webGL?

这些组件的渲染是通过，数据与html5描述实现的：
核心技术应该是：前端html5模板渲染引擎（Lexer)，数据绑定，VirtualDom

说到底前端技术就是，按平台分android, ios, browser等等。UI描述xml, html5,...
用户端程序执行语言(js, java)，通信技术(tcp, http)等等

node说白了coroutine, 线程替换技术，什么语言都有，不要过度神秘化

在 2017年6月4日 上午1:47，Yiak Wang Yi <yiak.wy@gmail.com>写道：

...

👎 4

yiakwy commented on 4 Jun

我告诉你，要想做后端，必须了解docker, 微服务架构，MAPreduce, 你js MVC写得再好也没用。这就叫技术问题。

在 2017年6月4日 上午1:55，Yiak Wang Yi <yiak.wy@gmail.com>写道：

...

👍 2 👎 3

calidion commented on 6 Jun • edited

@atian25

有意思啊。

还偷偷的点 👎

你有道理可拿出来说说？

本来不太想说你，毕竟你说我眼界低，我也承认。

何况我还做了好几个打你们这群人脸的项目。

但是你为了证明你的眼界高偷了我的框架名(egg)，然后还写的性能比我的框架低很多，理念上还是抱着基本的MVC这样落后的观念。写出来了比垃圾回收站还牛的eggjs，什么都往上堆，就差将整个npm包装进eggjs了。

这就是你所谓的有眼界吧？眼界高就是去做低水平重复建设的事情？

看来我的世界观也得改一下了。

据我了解的眼界高的项目比如seajs，再比如eggjs:)

估计你们这些人都是眼界高一派的老手。

当然你们眼界高肯定跟你们公司是无关的，是真高，放在那里都是一座高山。

因为我的框架处理每个请求的时间比eggjs低3倍多，所以我的眼界确实不行。

希望eggjs可以向眼界高处走，将处理时间再提高点。哈哈。

对了差点忘记了，为了证明你们眼界高，你们一定不要忘记将所有你们知道的设计模式引入到eggjs中去。

否则就会跟我的vig框架一样，没有任何可以证明自己眼界高的名词了。

会很丢人的，会被认为眼界低:)

不过eggjs终于让我理解了眼界高是什么意思：就是低水平重复建设:)

👍 5 👎 2

zhangzhenhua92 commented on 8 Jul

纯粹来涨涨知识，简直太棒了。

Jandaes commented on 10 Jul

看了对web分工合作更加详细了解了、但是这样的话、java后台是不是只要提供webservice的RESTful 就行了！那什么重定向、转发的都不用吗？

calidion commented on 10 Jul

@Jandaes

少看阿里一些技术人员的奇谈怪论，特别是阿里早期技术员工的言论。

如果你知道了阿里某些表演系技术人员连中间件是什么都搞不清楚的话。

你就应该对此不会奇怪。

https://t1bao.com/thread/visit/57

👎 1

chenbin92 referenced this issue in **chenbin92/blog** on 16 Jul

你可能不需要 **Vuex** #1 Open

jedyang commented 7 days ago

作为一个刚接触前端的后端开发，仔细看了文章和评论，长知识了。感谢

Summer120 commented 4 days ago

非常有启发

lonelycheng commented a day ago

如果可以把前端到后端的每一项技能要求都减少一些，那么我们就可以称自己为全栈了。

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Comment