大型网站之网站静态化（javascriptMVC架构） - Robin Hu的专栏 - 博客频道 - CSDN

原文：http://blog.jobbole.com/84532/
http://blog.jobbole.com/84590/
http://blog.jobbole.com/84595/

一、简介

要求实现网站静态化的一个重要点就是前后端分离技术。我们要深刻理解前后端分离技术有一个重要的前提，那就是要把前后端分离技术认为是传统的web应用里的MVC设计模式的进一步演进。那么我们首先来看看MVC的定义，下面的内容摘录于维基百科的解释，具体如下：

MVC 是一种使用 MVC（Model View Controller 模型-视图-控制器）设计创建 Web 应用程序的模式：[1]

Model（模型）表示应用程序核心（比如数据库记录列表）。

View（视图）显示数据（数据库记录）。

Controller（控制器）处理输入（写入数据库记录）。

MVC 模式同时提供了对 HTML、CSS 和 JavaScript 的完全控制。

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。

通常模型对象负责在数据库中存取数据。

View（视图）是应用程序中处理数据显示的部分。

通常视图是依据模型数据创建的。

Controller（控制器）是应用程序中处理用户交互的部分。

通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

各类用于 Web应用开发的语言里都有属于自己的MVC框架，例如本人最熟悉的服务端语言Java里就有大名鼎鼎的struts2，springMVC的MVC应用 框架。 像struts2和springMVC这样的框架做了太多浏览器本身就可以完成的工作，例如：页面的渲染操作，因为服务端抢了浏览器端的部分工作，这其实也就等于限制了web前端技术的深入运用，像很多前端的优化技术以及很多提升用户体验的技术就很难派上用场，之所以产生这些 问题，我认为传统的MVC框架本质其实是一个服务端的MVC框架，虽然MVC设计模式里的V即View视图层是想把界面开发工作专业化，让界面设计人员能专心于界面开发，但是传统的MVC框架下的View层的本质却是一个不折不扣的服务端技术。

我们以java的web开发里jsp为例，JSP全名为Java Server Pages，中文名叫java服务器页面，其根本是一个简化的Servlet设计，它是java里动态网页的技术标准， 这就说明jsp虽然看起来像html，其实它并不是真正的html，它需要被java的web容器进行解析转化为浏览器可以解析的html页面，然后通过 网络传输到浏览器后，浏览器才能正确的展示这个jsp页面，其他web开发语言里都有类似的动态网页技术标准，但是不管什么语言的动态网页技术标准，我们 使用它时候就是让web前端技术被服务端技术所绑架，这也就是为什么每个招聘web前端工程师的岗位都要问你是否会java，PHP语言的源头。但是随着 互联网的大发展，对web前端的要求是越来越专业化，web前端本身所包含的技术难度已经不亚于任何一个

服务端语言开发难度，因此我们需要web前端更高 的专业化，而不希望web前端工程师被服务端技术束缚的更多而限制了自身能力的发展，这就导致前后端分离技术的出现。

不过前后端分离技术的第一阶段倒不是从改变view层即视图层开始的，而是从连接客户端和服务端的C层即控制层开始的，控制层既要作用于客户端又要作用于服务端， 如果一个功能页面是一个程序员从浏览器端一直写到模型层，控制层也就不是什么问题了，但是如果当我们想按MVC的设计思想，让界面开发人员专注于页面开 发，服务端开发人员专注于服务端开发，那么这个时候控制层的归属问题就显的非常重要了。在传统的MVC框架里，因为M层和C层是使用同样的语言体系，因此 我们很自然会把M层和C层的开发工作都交由服务端开发人员完成，这个决定无可厚非，但是传统的MVC框架里V层和C层其本质也是同一个技术体系下的（例如 java的web开发里的jsp本质就是个servlet），因此V层和C层也是紧耦合的，因此界面开发人员开发页面时候如何没有C层支撑，那么这个页面 其实是根本跑不起来的，如果前端开发人员这时候跑去写写C层即控制层的代码，这就打破了原有的横向分工，这个时候控制层的编码工作就会变得混乱而难以控 制，看到这里有人一定会说既然控制层是属于服务端的，那么前端技术人员就等等服务端的开发进度，再不行就自己写个mock模拟下服务端的控制层，听到这种 建议，我相信不管是前端的还是服务端的技术人员都会头脑发麻，第一反应就是这不是自找麻烦啊，还不如一个人全部搞定算了。由此第一阶段的前后端分离技术方 案出现了，这个方案需要解决的问题就是如何能让web前端技术人员和web服务端技术人员协同起来工作，合理的分工，换句话说就是按web前端和web服 务端角度如何能横向的分解web的开发工作。

二、服务端驱动的前后端分离

前后端分离的第一阶段需要解决问题的核心就是控制层的归属问题，从技术角度而言就是控制层到底是应该和视图层解耦比较合理还是跟模型层解耦比较合理的问题。那么我们这里先回顾下MVC设计模式里对控制层的定义，维基百科里的定义是：

（控制器 Controller）- （控制器）是应用程序中处理用户交互的部分。通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

在web开发里，我们发现控制层以沟通视图层和模型层的角度而言，控制层其实主要完成三项具体的工作，它们分别是：

工作一：控制层起到一个路由的作用。客户端请求到达控制层后，控制层根据请求内容将请求路由到服务端某个模型层进行处理，模型层将请求处理完毕后，会把响应结果返回给控制层，控制层在根据响应信息路由到特定的页面。

工作二：控制层起到一个报文信息格式转化的作用。这里以java的web开发为例，浏览器的数据都是以http报文形式发送给服务端，而控制层就是将http报文信息解析成java的对象，当然也可以是java的基本数据类型，然后控制层把解析好的信息传递给模型层进行处理。

工作三：传统的MVC框架里，控制层其实深入参入到了页面渲染的操作。在 java的web开发

里的控制层不管如何被包装，其本质就是一个servlet，而jsp页面本质也是个serlvet，因此我们可以这么理解 jsp，jsp就是以页面开发的方式写java，而servlet就是以java的方式写页面，所以我们可以在servlet里以文件流的方式输出页面， 也可以让servlet跳转到jsp页面。

由上面的 论述里我们发现，其实传统MVC框架里控制层和模型层的联系方式相对很简单的，它们的联系主要是路由和报文格式的转化上，而控制层与视图层的联系除此之外，还多了一个页面渲染，而页面渲染本身应该是属于浏览器的技术范畴，是浏览器技术不可分割的一部分，也是我上面内容里诟病传统MVC框架问题所在，如果控制层承担了页面渲染工作，那么控制层和视图层的耦合度就变得非常高，要想将其解耦是十分困难，一般只有我们打破了现有MVC框架的技术体系才能完成，相比之 下，控制层与模型层的解耦就显得容易多了。那么控制层与模型层如何解耦呢？具体如下：

首先我们来解决下报文格式转化的问题，这个技术方案很简单就是借鉴http统一报文格式的特点，我们为控制层和模型层定义一套统一的报文格式，例如我们定义控制层 和模型层都以map的数据类型进行数据传递，这个map里有个专门的字段用来定义被路由到的模型接口信息，有个字段专门存储需要传递的数据，具体的设计方 案可以根据实际的业务需要来设计。

然后就是路由的问题了，在解决报文格式转化问题的论述里我讲到要在统一报文格式里专门定义一个字段用来存储该数据到底路由到哪个模型进行处理，不过这个字段并不能完全解决路由问题，因此我们需要模型层对控制层提供一个统一的接口，任何控制层与模型层的沟通都通过这个统一接口来完成，只不过不同请求报文组装的内容 不一样而已，而这个接口还有个重要职责就是解析报文里的路由信息，让请求能被正确的路由到对应的模型接口所处理。当然这个接口的返回值最好也是一个统一的 报文格式，这样控制层解析模型层的返回数据也会便利的多了。

由上所述，我们发现第一阶段的前后端分离工作控制层应该归属于web前端，这么做更加合理，也更加容易实现。

这个方案进一步演化一下，我们可以把控制层和视图层独立成一个web应用，模型层也独立成一个web应用，两个web应用之间通过远程调用方式进行沟通，关于这个方案可以参考《我设计的网站的分布式架构》。

这个进化 版的方案增加了系统开发的难度，因为我们需要增加网络通信的编程以及远程调用的实现，更麻烦的是我们还需要进行复杂的多线程编程，既然增加了开发的难度为 什么我还要这么做呢？首先我们通过应用分层，可以动态的调节web前端和web服务端的负载压力，还可以在模型层之前提供一道安全屏障，不过被服务端绑架 的web前端在提升整个web应用负载能力这块还是很有限的，其实这种做法的最大好处就是利于SOA框架的设计，也就是说这种架构我们可以为服务端的 SOA化提供有力的保障，因为控制层和模型层的解耦，可以让模型层真正做到专注于业务，而不会再发生那种把业务逻辑写到控制层的问题了从而降低代码的健壮 性。

三、前端Web驱动的前后端分离

上文的前后端分离方案其实是服务端驱动的前后端分离方案，它的实现手段又是从服务端的MVC架构体系演化而来，因此该方案最大的问题就是它并没有从根本上改变web前端从属于服务端的被动局面。那么问题来了，有没有以web前端为驱动的前后端分离方案呢，该方案能让web前端的能力获得更大的释放了？答案是绝对有。本篇就要讲讲以web前端驱动的前后端分离方案。

首先要提的就是javascriptMVC，下面我摘抄的是维基百科里对javascriptMVC的解释，具体如下：

首先是简介：

JavaScriptMVC 是一套开放源代码的多样化互联网应用程序框架，以 jQuery 与 OpenAJAX 为基础。JavaScriptMVC 利用 MVC 架构与工具扩展这些函式库，以便开发与测试。由于 JavaScriptMVC 不需要任何服务器端的配合，因此它可以和任何的网站服务接口与编程语言整合，如 ASP.NET、Java、Perl、php、Python 或 Ruby。

接下来是历史：

JavaScriptMVC 的第一个版本是在2008年5月释出。稳定版的 JavaScriptMVC 2.0 在2009年6月释出，并以 jquery 为基础。主要开发目标为维持程式码的简短和专注在它独特的功能上。3.0版本在2010年12月释出。而从 JavaScriptMVC 中所独立出来的 MVC 架构“CanJS”则在2012年4月释出。

从维基百科里的解释我们会发现如下启示，它们分别如下：

启示一：javascriptMVC是一个应用框架的名字，这和jQuery的命名是一样的，所以这里我要声明一下，本系列里的javascriptMVC不是指代这个框架，而是指代的是使用javascript语言实现出的一类的web前端的MVC框架，本系列后面的javascriptMVC和前端MVC的含义是一致的。

启示二：从javascriptMVC历史里我们可以看到第一版的javascriptMVC产生于2008年，这个历史要远早于nodejs出现的时间，这说明了前端的MVC并不是因为nodejs的出现而产生的，应该是nodejs推动了前端的MVC框架的应用和普及。

启示三：维基百科里有一段解释：

由于 JavaScriptMVC 不需要任何服务器端的配合，因此它可以和任何的网站服务接口与编程语言整合，如 ASP.net、Java、Perl、PHP、python 或 Ruby。

这段话说明了前端MVC的一个很重要的特点就是前端MVC可以摆脱服务端语言的束缚做到真正的独立，同时前端MVC又可以和任何服务端语言进行整合，大家可以试想下如果我们开发的web应用前端达到了前端MVC的程度，那么公司在招聘web前端工程师的时候就不在会问你“你会java吗？”或者“你会php吗？”假如这个前端工程师所会的服务端语言能力和公司不匹配，面试官也不会再犹豫和摇头了。

启示三同时还隐含了一个问题，为什么好的前端MVC框架可以做到和任何服务端语言配合呢？这个解决手段之一我在前文中的第一阶段前后端分离方案里就提到了，那就是解决报文格式的统一和交互接口的统一的技术手段，只有这样前端MVC和服务端的灵活对接就不会再是问题了。但是仅仅这个手段还是远远不够的，我们要达到这个需求还需要解决一个问题，这个问题就是要把服务端MVC霸占web前端的工作也要抢回来。那如何抢呢？

服务端MVC的视图层技术例如java里的jsp技术，这个技术是将html和java代码整合的技术，java的web容器把jsp解析完毕后最终生成为html文件发送给浏览器，浏览器在解析这个html将最终效果展示给用户。那么我们要抢回服务端霸占的web前端的工作我们就得分析下这些动态页面技术到底做了哪些事情特别是侵占web前端的事情。

这里首先我们要谈谈服务端在动态页面里的作用，其实服务端为动态页面作用很单一就是提供了网站需要展示的数据而已，服务端是不会创造一个新页面的。服务端提供的数据的类型也是很统一，要不就是服务端语言提供的基本数据类型例如：字符、数字、日期等等，要不就是复杂点的数据类型例如数组、列表、键值对等等，不过归属服务端的动态页面还需要服务端语言帮助做一件事情，那就是把服务端提供的数据整合到页面里，最终产生一个浏览器可以解析的html网页，这个操作无非就是使用服务端语言可以构造文件的能力构建一个符合要求的html文件而已。不过一个页面里需要动态变化的往往只是其中一部分，所以做服务端的动态页面开发时候我们可以直接写html代码，这些html代码就等于在构造页面展示的模板而已，而模板的空白处则是使用服务端数据填充，因此在java的web开发里视图层技术延生出了velocity，freemark这样的技术，我们将其称之为模板语言的由来。由此可见，服务端MVC框架里抢夺的web前端的工作就是抢占了构建html模板的工作，那么我们在设计web前端的

MVC框架时候对于和服务端对接这块只需要让服务端保持提供数据的特性即可。从这些论述里我们发现了，其实前端MVC框架要解决的核心问题应该有这两个,它们分别是：

核心问题一：让模板技术交由浏览器来做，让服务端只提供单纯的数据服务。

核心问题二：模板技术交由浏览器来承担，那么页面的动态性体现也就是根据不同的服务端数据进行页面部分刷新来完成的。

而这两个核心问题解决办法那就是使用ajax技术，ajax技术天生就符合解决这些问题的技术手段了。

要让web前端承担模板技术，就得使用javascript的模板技术，时下javascript的模板技术可谓是百花齐放，百家争鸣，很多朋友曾为这些技术称奇，其实探求它的本源无非就是用javascript为基础实现了个jsp，velocity而已，如果有朋友还没接触过javascript模板技术，可以在百度里搜索下【javascript模板引擎】，本文这里就不展开谈论了。

前端的MVC不应该等于单页面开发，前端MVC也不是把ajajx用到极致，根据实际业务场景，我们需要适当的把同步请求和异步请求结合起来。如果前端MVC里包含了更多同步请求，那么对于MVC里的C层即控制层就会有更高的要求。

四、NodeJs实现的前后端分离

前端Web驱动的前后端分离,在带来好处的同时，也带来一些问题。主要问题如下：

问题一：前端MVC让web前端的技术难度和架构难度成指数级上升，而javascript语言天生有着自己设计的缺陷，这个缺陷在写大规模复杂应用时候就显得尤为突出，例如：javascript没有模块化管理，javascript面向对象的实现难度，所以前端MVC的应用可能会变相的提升企业的技术成本和开发成本，当然很多新的技术手段能解决javascript固有的缺陷，对这些新技术有个更大的问题就是“你会吗“，不会的话首先要解决会的问题，这也是个成本问题。

问题二：当前端真的越来越独立于服务端后，这会导致服务端一些可以优化web前端的重要技术就很难实现了，例如网站静态化系列里讲到了缓存运用，CDN的运用就很难达到预期效果，或者根本没法使用，因为这些技术的根基都是认为网站动态性是由服务端发生的，而客户端霸占了动态性，那么这些技术的作用就被限制住了。

由此我可以下个结论：如果前后端分离方案是以浏览器和服务器角度来划分并不是最好的前端分离方案，那么前后端分离方案还有没有新的解决思路了？这个真的有，那就是nodejs参入的前后端分离方案。

其实前后端分离的驱动永远都是前端强于服务端，而前后端分离的重要目的也是要给web前端创造一个更加干净的开发环境，那么写的代码是否是在浏览器上跑还是在服务端上跑这个并不是太重要，所以引入nodejs，就是让服务端也能跑javascript代码并不会是让人无法接受的事情，回到前后端分离方案里以服务端驱动的前后端分离方案，我曾说过这个方案能获得服务端开发人员更多的掌声，我相信这个掌声不会是服务端为前端的喝彩，而是服务端终于从web前端解脱出来了，这样服务端运用更加高级的SOA技术就成为了可能，那么我们把web前端的控制层使用nodejs替代，这么一来我们既可以继承所有传统MVC框架的优点，同时也达到以前后端分离的根本的问题就是为web前端创造一个很干净的开发环境问题，那么我们在前端MVC框架使用时候遇到的问题都会很好的被解决。

Nodejs的运用让动态网站的动态性再度停留在了服务端，那么我前面讲的那么多网站静态化技术就可以和前后端分离方案很好的融合了，因此本篇先不具体讨论nodejs做前后端分离的实现手段了，在下篇讲从网站静态化角度重新审视前后端分离方案时候一起讲解，这么做会更加符合本系列的主题。

为什么nodejs技术可以突破传统服务端技术的包围，因为nodejs可以让前后端达到更高程度的分离，从而让前后端各自发挥自己的优势，很有意思的是，虽然nodejs技术属于服务端范畴，但是它却是前端工程师驱动来普及的，这绝对是web前端逆袭啊。

假如我们的网站控制层相对比较简单，好了，这时候我们跟领导或老板说“现在很流行前后端分离， 我们项目也使用下前后端分离技术”，领导或老板一听可能会为之一振，那么就会问你”那么该怎么做了”，你这时对他说“首先把控制层用nodejs重写 下”，领导或老板听到这个回答他会同意你这么干嘛？一个不会给网站增加任何新功能，同时不能很直接有效的提升网站的性能，而且执行它还会有很大风险的方 案，头儿们会同意吗？好了，假如你终于找到合理理由说服头儿们，那么如果我们的网站规模已经很大，控制层已经演变成了网关项目，控制层本身已经巨复杂了， 你敢用nodejs重写一遍网关项目吗？所以说吧nodejs直接当做控制层，其实实践起来困难重重，而且nodejs完全承担控制层，它的性能，它能否 很好的运用于集群开发这都是很难把控的问题。分析到

这里，我们似乎又进入了死胡同了，那如何来破这个局呢？

上面的问题只 是反映出整个网站MVC里的控制层其实还有部分功能是和服务端的模型层紧耦合的，因此要解决这个问题就是把传统的控制层再细分一下，属于前端的部分划分给 web前端作为web前端的控制层，属于服务端的部分任然留给服务端，这么拆分后，当我们引入了以nodejs为基础的前后端分离方案，服务端的控制层改 造无非就是去掉页面路由，页面渲染，再修改下返回数据格式即可，因为不用修改服务端的业务代码，其代价是很低的，头儿们也很容易接受这样的方案，并支持我 们大胆去尝试新技术。

服务端网站静 态化技术SSI和ESI，主要是根据动静分离策略把网页不会经常变化的模板进行缓存，然后在静态资源服务器位置整合动静资源，如果我们使用nodejs只 是简单替换原来的控制层，那么这些策略其实还是有问题的，那么怎样做可以让nodejs兼容SSI和ESI了？这里我列举个实际的案例，nodejs有一 个模板语言叫做jade，nodejs里还有个技术叫做handlebarsjs，其中handlebarsjs和struts的标签类似，它可以处理一 些简单的业务逻辑，我们开发时候使用jade编写页面的模板，使用handlebarsjs让动态数据和模板进行整合，项目发布时候，使用像grunt这 样的项目管理工具编译项目，jade文件变成html文件，而handlebarsjs则会转化为javascript代码，这样我们就可以把生成的 html文件在服务端进行有效缓存，而handlebars生成的javascript文件负责整合动静数据，这样nodejs就可以达到兼容SSI和 ESI的作用了。

不过引入 nodejs会让网站处理请求的过程里增加一个环节，这样可能会导致部分性能的损失，但是我上面的实例却能有另外的方式规避这个问题，因为nodejs的 代码是用javascript语言编写的，那么这个代码是可以运行在浏览器上的，那么这就会产生了一个处理手法，那就是我们在生产部署时候其实不需要部署 nodejs的，我们把静态模板就缓存在服务端或者推送到CDN上，然后handlebarsjs生成的js代码就让它传送到浏览器端，因为这个js代码 生成后基本不会变化，浏览器可以缓存它，当然CDN或静态资源服务器也可以缓存它，其实它在浏览器运行时候变化无非就是获取一次服务端数据而已。这么一 来，生产上的web前端又转变成了前端MVC的形式，还把动静整合的事情交由了浏览器来完成，这不仅是兼顾的网站静态化要求，还让动静整合推到了更加靠前 的浏览器端，这不是达到了一个双赢的效果了嘛。不过这个做法总是有点不伦不类的意味，假如我们真的想把nodejs引入到应用生产的网络架构里，我们不希望无端的增加请求处理环节，那么最好是让nodejs服务器替换某个部分。按照这个思路思考，那么我觉得nodejs在生产的引入最好是和反向代理服务器相关，最简单的方式就是让nodejs和反向代理服务区一一对应，这样就可以很好的降低引入nodejs带来的问题，当然复杂点的就是反向代理集群对应的应用服务器应该是nodejs的应用服务器，而不是用来做业务处理的业务级别的应用服务器。关于反向代理的更多内容请参考《大型网站之网站静态化（反向代理）》