**前言：分离模式**

　　对前后端分离研究了一段时间，恰逢公司有一个大项目决定尝试使用前后端分离模式进行，便参与其中。该项目从2016年初立项至今，平平稳稳得度过，但也涌现出越来越多的问题，绝对不是说前后端分离模式不好，而是很多公司在尝试前后端分离的时候没有做好充分得准备。

　　网上对前后端分离介绍的文章已经屡见不鲜，接下来本人用一点粗浅的言语也谈谈这块，献丑了。

**为什么要分离？**

　　如果只问“前后端分离的意义大么？”这是废话，因为从软件架构的角度 Web 的前后端从一开始不就一直是分离的么，而且 browser、server 可能将永远分离下去。

　　为了了解这个问题，我们有必要先了解一下 Web的研发模式演变，关于这个题材，下面这篇博文说得不错，这边就不做搬运工了。

　　我们不能“为了分离而分离”，而应该“为了真正理解web开发、为了更好完成需求而分离”。

**前后端分离的误区？**

**1、前端人员配备是否充足？**

　　由于所在公司以往项目采用传统开发风格，即以后端MVC为主的开发模式，前端人员仅仅提供静态html页面，其余工作皆由后端开发人员完成。采用前后端分离模式可以减后台负担，加快研发效率，当然，前提是前端能做好的话。以往只需要提供静态页面的前端人员，在前后端分离模式中要负责项目的view+controller部分，即除了静态页面，还需要负责页面的所有交互代码、以及nodejs与视图层以及后端API的交互工作，无疑增加了前端人员的学习成本，在没有足够知识和人才储备的情况下，只能让前端人员加班加点。结果是大量前端人员离职（PS：做这么多事，工资总得加吧！）

**2、前后端职责分配？**

　　很多公司认为采用前后端分离之后，前后端只需要通过指定API进行交互即可，前端负责页面渲染，Nodejs负责路由分配，后端提供API。忽视了大量关键工作，职责分配和细节处理没有相应文档规定，缓存机制、图片上传下载、数据校验、语言国际化等等并没有出具相应信息。另外，大量忽视了nodejs层的作用，仅仅把nodejs当成一个路由中转，这一方面也是对nodejs技术的不熟悉导致的，其实nodejs能负责很多事，除了复杂业务逻辑处理和数据操作由Java 负责，大量工作完全可以在nodejs层处理。（PS：还是基础不够导致的！）

**3、后端API是否Restful风格？**

　　很多公司采用了前后端分离模式后，后端API仍然采用以往的传统风格，这是不合理的，Restful风格的API应该是前后端分离的最佳实践。ResultFul推荐每个URL能操作具体的资源,而且能准确描述服务器对资源的处理动作，通常服务器对资源支持get/post/put/delete/等，用来实现资源的增删改查。前后端分离的话，这些api-url是对接的桥梁，采用resultFul接口地址含义才更清晰、见名知意。（PS：用了Spring4.x 竟然还不用rest风格，说不过去啊）

**4、前后端协作模式？**

　　前后端分离后，无论是API接口的对接还是测试工作，都涉及到前后端人员的沟通，很多公司采用前后端分离后，前后端协作模式配合力度底，互相等待，开发效率低下，反而不如传统的开发模式。例如：当后端 API 没有编写完成时，前端无法进行调试，这就导致了前端会被后端阻塞的情况。其实像这种互相等待的模式需要改进， Mock Server 可能可以解决一些问题。

**如何前后端分离？**

　　怎么做前后端分离？大方向就是

　　后端专注于：后端控制层（Restful API） & 服务层 & 数据访问层；

　　前端专注于：前端控制层（Nodejs） & 视图层

　　本人认为的前后端分离模式应该是这样，当然这不一定正确：

　　1、项目设计阶段，前后端架构负责人将项目整体进行分析，讨论并确定API风格、职责分配、开发协助模式，确定人员配备；设计确定后，前后端人员共同制定开发接口。

　　2、项目开发阶段，前后端分离是各自分工，协同敏捷开发，后端提供Restful API，并给出详细文档说明，前端人员进行页面渲染前台的任务是发送API请(GET,PUT,POST,DELETE等)获取数据（json，xml）后渲染页面。

　　3、项目测试阶段，API完成之前，前端人员会使用mock server进行模拟测试，后端人员采用junit进行API单元测试，不用互相等待；API完成之后，前后端再对接测试一下就可以了，当然并不是所有的接口都可以提前定义，有一些是在开发过程中进行调整的。

　　4、项目部署阶段，利用nginx 做反向代理，即Java + nodejs + nginx 方式进行。

**编后语**

　　从经典的JSP+Servlet+JavaBean的MVC时代，到SSM（Spring + SpringMVC + Mybatis）和SSH（Spring + Struts + Hibernate）的Java 框架时代，再到前端框架（KnockoutJS、AngularJS、vueJS、ReactJS）为主的MV\*时代，然后是Nodejs引领的全栈时代，技术和架构一直都在进步。虽然“基于NodeJS的全栈式开发”模式很让人兴奋，但是把基于Node的全栈开发变成一个稳定，让大家都能接受的东西还有很多路要走。创新之路不会止步，无论是前后端分离模式还是其他模式，都是为了更方便得解决需求，但它们都只是一个“中转站”。