前端架构

缘由:

作为一个长期从事前端开发者, 对于前端架构一直没有找到所谓官方的定义, 也暂时未找到一个自己觉得相对完美的解说.所以如何去做好一个架构师,如何复制一个项目架构过程这里也是对各大博主对前端架构的解释的做个小结, 方便自己以后查阅.

存在原因:

实际上很多公司可能不会有前端架构师这个职位,但也不是说没有了这个职位,相关事情就没有人来做, 相反它的责任一般是由前端组长或经理来承担.也就是说项目中一个人有一个人的安排, 多个人有多个人的安排, 所以这要看公司规模和项目规模.

所以一个公司开始有个架构师的预算, 就说明公司的代码管理这块有问题需要人来处理

职位划分:

前端架构师与前端经理(主管)皆为管理,前端经理管理人和任务, 架构则负责管理机器和代码.

之前的前端简单?现在复杂了?

之前的简单复杂度不高, 现在 组件化了, html拆分各种模板, js 拆分成各种模块, css也通过less or sass 这种可编译式的语言了, 代码自动化处理了 压缩,混淆, 合并, 发布..这些制度化, 流程化的培训体系

几个误区:

1. 架构师要参与开发
2. 架构不是一蹴而就的事, 在满足业务需求的情况下不断地完善和演进

什么是好的架构:

提高代码的可读性: 清晰的目录划分, 统一的命名规范

提高代码的可维护性: 低耦合，高内聚 (

1. 修改一个组件导致另一组件的变化
2. 新增需求、更改需求、修正bug 造成变化

)

提高代码的可扩展性:

便于协同:

自动化: 打包, 压缩, 混淆, 测试, 发布

知识储备:

前端开发: PC端, 移动端, 微信端, 桌面端, 服务端, 浏览器插件

PC端: Web

移动端: 纯Web | 混合App| 原生App| Flutter,

服务端: NodeJS,

微信端: 小程序, 公众号

桌面端Electron | NW

浏览器插件

前端框架:

Spa: react , vue, angular

其它: jquery , zepto, seajs, require.js...

打包工具: webpack, gulp, grunt,

架构过程:

1 项目信息收集(做事的意义)

1）这个项目的背景（为什么要做？为了解决什么问题？紧急程度如何？重要性？）

2）这个项目和现有项目的关联和它所在的位置（这个是为了确定边界和影响）

3）这项目的用户群体（是toB还是toC？是否有SEO要求？手机端还是PC端？使用设备及浏览器情况？访问高峰期时间？用户规模多大？）

4）项目涉及的业务的了解（特定业务知识储备、竞品了解、可能遇到的特定难题）

这个过程主要是为了建立起一个全局观，也引导架构最终所确定的形态，所以不要急于写代码，可以在脑海里先回顾下这些信息有没有了解充足，再有所计划的开展下一步。

1. 技术选型

1）PC端还是手机端(是否需要pc与mobile自适应, 数据是否一样, 布局是否一样)

2）toB还是toC

3）用户规模

4）浏览器支持情况(兼容IE9以上说明可以使用主流框架)

5）项目是否需要支持SEO(ssr, 骨架屏)

6）团队成员对该技术（或框架）的掌握情况：是否有人可以全局把控、学习成本是否可以接受等

7）该技术能否可以满足业务需求

8）该技术（框架）本身的稳定性，它的社区、维护者等

没有说哪个项目一定要选择哪个框架, 只是说我们需要在这些问题时做个利弊权衡

1. 项目目录结构

为了让项目显得有逻辑性, 便于管理源代码, 目录划分可以降低新手对项目理解成本, 常见的目录划分如 build , server, assets, src, dist 等

1. 自动化构建

1）提高开发效率：热加载、开发和生产代码分离

2）优化性能：代码合并压缩、文件版本号、按需加载、图片优化

3）提高代码质量：模块化、ES6+Babel编译、css预处理、eslint代码检查、无用代码片段过滤

以上这些也有较成熟的webpack配置方案

5. 常见问题:

样式的基础库

自定义主题色

字体图标的引用

移动端屏幕适配

错误处理

按需加载

图片懒加载

多语言支持

跨域处理

工具类方法封装

路由处理

加载中状态处理

服务端渲染

登录状态的判断

不同环境的配置文件

增量打包发布

1. 架构的演进

架构师的工和内容参考:

参与前端项目的开发和优化，创建严格的技术标准和规范，追求高效自动的开发流程，对技术改进、性能优化、技术创新能够提供解决方案。作为技术领路人和布道师，能够带领着其它小伙伴朝向更高更强的目标前进, 具体步骤:

1. 持续关注技术的发展，掌握新技术的核心点，从中挑选适合我们自己的新技术；
2. 建立完善的技术标准和规范，并且能够监督和执行
3. 建立自动化流程，包括部署发布，规范检查，测试流程
4. 开发创造更多前端工具
5. 参与功能开发
6. 指导和培训工程师，促进团队成员的进步

Nodejs 接口层

痛点:

1. 前后端一般属于不同的部门，在工作中都会下意识地把麻烦推给对方，造成潜在的冲突风险。

2. 前后端的思维模式不同，造成相互不理解。

2.1在后端开发看来，接口层只是个二传手，没什么技术含量。

2.2在前端开来，后台给的数据不是多了，就是少了，很少有称心如意的。

由前端的人来写接口，更理解需求

同一个部门，接口调试更加方便

前端多样性，有iOS，Android，小程序，PC，各种端；经过这一层包装之后，后端具体的服务，只要对接一层就可以了。

Java，高并发要求比较高。Node.js，本身就是从异步高并发角度去设计的，比较适合做接口层

客户端的JS，PC端的JS，Node.js都可以部署在同一服务器上，解决同源问题