

引言

WEB前端技术早已今非昔比，可谓是大前端。从网页上的炫酷效果，到页面的布局，再到服务端模板渲染和数据接口功能，WEB前端技术都可以轻松搞定。甚至利用JavaScript可以编写很多跨界的项目，如游戏开发（Cocos2d\Unity3D）、机器学习（ConvNetJS）、跨平台应用（ReactNative）等等。

如今jQuery已成是一个优秀的古老的存在，前端技术日新月异，如果你是前端开发者，不知

道：React、Vue、Angular、Webpack、Gulp、Sass、LESS、PostCSS、NodeJS、Koa、Express、Mocha、Yarn、Redux、Middleware、ES6/7、TS、Coffee、Babel等等这些词，那你真该好好学习一番了(说的不全，就这个意思吧)。

就本文而言，单说说网站开发这块的事儿，看看2017年的WEB技术能做到什么程度。

技术的初衷就是解决问题

问题的存在与技术的存在不像“先有鸡还是先有蛋”的问题那么纠结，这必然是先有问题，再有对应的技术解决此问题的。而问题的产生往往是因为我们追逐着完美，想要更优秀。在网站开发这个领域同样如此，网页是人机交互的入口，我们追求的正是响应的速度、展现的效果和使用的体验。

那么问题就来了（先只抛砖，后面有讲解）

一、如何让网页响应的速度更快呢？

现有情况

1. 传统的多页面（服务端渲染HTML）方式

把公共部分的JS/CSS引用到页面里，首次加载会下载额外的资源文件。切换页面重定向的时候，整体页面刷新，很多资源被2次下载，并且有一

段时间页面空白。

问题：下载了额外的资源文件，页面重定向会有空白间断。

2. 单页面WEB应用（SPA）方式

把页面的基础HTML壳子和页面所需要的JS/CSS是先下载到浏览器端，再进行HTML渲染，整个网页必须在引用的JS文件下载后，n 毫秒后显示。

（即使做了AMD方式的分包，可以优化节省下载时间，但无法根本解决页面展现依赖JS下载完成问题）

问题：网页的整体显示依赖于JS是文件下载时间。

理想情况

根据当前URL请求，返回对应的（完整内容）HTML和（当前页面所需要的）CSS以及（基础功能和当前业务）JS。

其中CSS放到页内，与HTML一同下载，会省了1条下载线程让给其他资源。（与传统方式不同的是CSS没有多余的部分）

其中JS放到最后加载，不影响网页首屏的显示。

后续的所有页面（与本站相关的网页）都是跟进用户的动作按需加载，做异步的JS处理。

后续加载的JS不会重复下载。

总结：在首次打开页面时候排除所有可能排除的下载内容，让网页瘦成一道闪电。

二、假如上面的理想情况实现了，后续的内容全部是JS加载渲染，那么SEO该怎么办？

现有情况

1. 传统的多页面（服务端渲染HTML）方式
有较好的SEO能力，可以在HTML渲染时候指定多处SEO优化内容。

2. 单页面WEB应用（SPA）方式
几乎丧失了多页面网站的SEO能力，也会有一些特定的方法解决，但效果跟多页面网站完全不是一个level的。

理想情况

我还没有更好的办法超越“多页面网站”的SEO能力，那就做到和它一样的level。

要做到2点：

- a) URL必须返回对应的HTML内容
- b) 前端开发的时候可以轻松设置SEO内容（如meta、title等）

三、假如上面的理想情况实现了，那么作为一家国际化的公司多语言是个必不可少的功能，该如何实现呢？

中小型项目

一般会编写国际化的语言变量文件，根据浏览器的设置或者用户的选择，调用不同的语言变量文件。目录结构如下：

```
--i18n
```

```
|
+---zh.js
|
+---en.js
```

大型项目

与中小型项目不同的是，大型项目有较多的语言变量，如果一次性加载导致速度慢，而且很多当时用不到的地方会产生流量的浪费。这时候我们会做拆分，根据功能分支，做按需加载。目录结构如下：

```
--i18n
|
+---feature-1
|   |
|   +---zh.js
|   |
|   +---en.js
|
+---feature-2
|
+---zh.js
|
+---en.js
```

另外，还一种方式跟据语言包种类，用Webpack根据语音种类，打出多套程序代码进行部署，比如要支持3个语言，那么就生成3套部署代码，线上跑3个网站程序。这种方式是性能最好的。

四、假如上面的多语言也实现了，公司做Android平台的项目那么多，我们前端同学能做什么呢？

[PWA 点击了解](#)

PWA是Google推出的渐进式Web应用程序，让H5在Android平台上增加了很多类似APP的功能，如推送、像应用一样放到桌面、离线使用等。

那么前端同学就可以利用Google的API来加强H5的体验。在实现的时候只是多了一些配置代码和对数据请求的特殊处理。

离线的核心技术就是ServiceWorker。从开发工作量的角度来说，并不会会有太多的额外工作量，自己可以基于NodeJS来实现一套工具自动完成这件事儿。

AMP 点击了解

AMP也是Google推出的增强网页体验和增强搜索能力的方案。有点类似小程序，严格要求安装他的语法格式编写网页。不一样的是他自带一个JS库，可以直接在浏览器上运行。

前端同学可以利用AMP的语法做出让用户更高效浏览的网站，特别适用于新闻类型的站点，Google搜索会对AMP实现的网站有专项推荐。从开发工作量的角度来说，会增加一些工作量，实现需要用到AMP特定的语法规范。

五、假如上面的多语言也实现了，那么我们的开发效率能不能更高？

说到这里，就不做比较了，前端攻城狮们或可以说“心灵手巧”，或可以说“懒到极致”，近几年是不断的：发明工具->重新发明工具->重新发明工具->重新...

工具会减少重复劳动的好东西，必不可少，这里我会从多个角度分析，如何提高前端开发效率。

跨职能联调

1. 与服务端开发同学联调

你是否工作中常听见：“他的接口还没好，等接口呢”、“页面写好了，就差套模板了”、

“服务端同学又改我代码”、“等服务端同学帮我搭建环境”...

这些问题可以通过与服务端同学的配合方式上解决，前后端开发完全分离，让服务端同学只开发接口，

更专注于数据业务逻辑和接口性能。让前端开发开发网站的非数据库部分，包括了服务端渲染和网页页面。

前端同学用NodeJS技术做服务端开发，代码部署到与服务端代码相同的机房，这样既不会影响网站速度，

又不会导致前后端开发互相影响。仅基于接口的联调会更容易测试、排除问题。（阿里早几年就已经开始这样做了）

2. 与测试同学的配合

基于上述的开发模式，接口可以加上单元测试，避免一些低级错误的发生。

前端同学自己的写的逻辑性代码也可以加上“单元测试”（有精力的情况下），

包括页面的交互上操作也可以加上“行为测试”（有精力的情况下）。这样在交QA同学的时候就会减少很多低级错误了。

团队开发有规范

1. 统一用ES、TS、Coffee等这些比较先进的语法

2. CSS使用LESS\Sass等

3. 统一编码风格，一个.eslintrc配置文件（ESLint）就可以搞定

4. 组件化开发：React\Vue\...

5. 适当的加上git hook，做code review

等等这些内容现在的前端开发比较常用，不细说了。

善用工具

1. WEB前端的工具应该是最丰富的了，代码检查、合并、压缩、混淆、SourceMap、CSS优化、各种任务预处理工具
前天的Grunt\昨天的Gulp\今天的Webpack
2. 开发的时候还有实时更新的工具，边写边更新页面：LiveReload、Webpack-dev-server、HMR
3. 图片资源的（视觉）无损压缩处理：Tinypng
4. 各种测试工具：Mocha\Karma\Jasmine
5. NodeJS进程管理工具：PM2
6. 不得不提的包管理工具：NPM(CNPM)\Yarn

总结：

1. 减少沟通成本，其实大多程序猿的coding速度还很快的。
2. 规范和工具都是围绕着“Don't repeat yourself!”这个思路来执行的。

把理想变成现实

上述的种种已实现并整理成框架，名为“super-project”，模板项目：[sp-boilerplate](#)

框架核心：[super-project](#)

引用技术：Koa+React+Redux+Webpack（可以使用类似的技术替换，如：Express+Vue+Vuex+Browserify）

下面简要简介了一些技术点实现原理，涉及内容较多，不能面面俱到，想深入了解的同学请点击链接查看源码，一探究竟，或直接找我沟通。

一、如何让网页响应的速度更快呢？让网页瘦成一道闪电

1. 准备好一份精简的HTML框架(跟进业务精简，酌情处理)。
2. 利用 React+React-Router 服务端渲染的能力（自研发Koa服务端渲染React功能中间件[super-project/ReactApp](#)），在服务端执行React代码，渲染出当URL请求对应的HTML片段。
3. 用样式收集器（自研发[sp-css-import](#)），收集到当前HTML片段里（React组件）需要的CSS。

样式收集器原理，利用组件化开发的方式，把每个组件对应的CSS独立出来，用于引用收集，利用React渲染机制，在用户访问页面时，把用到的组件的CSS注入到收集器中，收集器在分类、压缩CSS。在网页内运行期间，页面内部组件会根据用户的操作发生改变，组件使用次数会有全局的监听，根据页面内使用到的组件动态调整页内