总结与补充说明

恭喜你，学完了本小册。下面来总结下本小册的内容，并补充一些遗漏的内容。

总结

本小册主要带领大家从准备简历开始，逐步梳理技术面试知识点和非技术面试常考问题，最后介绍了一些谈 offer 之类的面试技巧。下面带领大家根据准备、技术面试、非技术面试和 HR 面试四部分，回顾一下每部分的要点。

准备阶段

简历准备：

1. 简历要求尽量平实，不要太花俏
2. 格式推荐 PDF
3. 内容包含：个人技能、项目经验和实习经验
4. 简历应该针对性来写
5. 简历提到的项目、技能都要仔细回想细节，挖掘可能出现的面试题

拿到面邀之后准备：

1. 开场问题：自我介绍、离职原因等
2. 了解面试官、了解公司和部门做的事情
3. 知识梳理推荐使用思维导图

技术面部分

集中梳理了 ECMAScript 基础、JS-Web-API、CSS 和 HTML、算法、浏览器和开发环境六大部分内容，并且就一些高频考题进行讲解。

非技术面试部分

主要从软技能和项目介绍两个部分来梳理。在软技能方面，介绍了工程师从业人员应该具有的软技能，并且通过几个面试真题介绍了怎么灵活应对面试官；在项目介绍小节，推荐按照项目背景、承担角色、项目收益和项目总结反思四步来介绍，并且继续推荐使用思维导图方式来梳理项目的细节。

HR 面

在小册最后，介绍了 HR 面试应该注意的问题，重点分享了作为一个 Web 前端工程师怎么对自己进行估值，然后跟 HR 进行沟通，拿到自己可以接受的 offer。

最后还介绍了一些面试注意事项，在面试整个流程中，太多主观因素，细节虽小也可能决定候选人面试的结果。

补充说明

本着通用性和面试门槛考虑的设计，本小册对于一些前端进阶和框架类的问题没有进行梳理，没有涉及的内容主要有：

1. Node.js部分
2. 类库：Zepto、jQuery、React、Vue 和 Angular 等
3. 移动开发

下面简单展开下上面的内容。

Node.js部分

Node.js 涉及的知识点比较多，而且比较偏后端和工具性，如果用 Node.js 来做 Server 服务，需要补充大量的后端知识和运维知识，这里帮助梳理下知识点：

- Node 开发环境
  - npm 操作
  - package.json
- Node 基础 API 考查

    - file system
    - Event
    - 网络
    - child process
- Node 重点和难点
    - 事件和异步理解
    - Steam 相关概念
    - Buffer 相关概念
    - domain
    - vm
    - cluster
    - 异常调优
- Server 相关
    - 库
        - Koa
        - Express
    - 数据库
        - MongoDB
        - MySQL
        - Redis
    - 运维部署
        - Nginx
        - 进程守候
        - 日志

Node 的出现让前端可以做的事情更多，除了做一些 Server 的工作以外，Node 在日常开发中可以做一些工具来提升效率，比如常见的前端构建工具目前都是 Node 来编写的，而我们在研发中，一些类似 Mock、本地 server、代码实时刷新之类的功能，都可以使用 Node 来自己实现。

前端框架（库）

jQuery 和 Zepto 分别是应用在 PC 和移动上面的库，大大降低了前端开发人员的门槛，很多前端工程师都是从写 jQuery 代码开始的。jQuery 和 Zepto 这两个库对外的 API 都是相同的。在面试的时候可能会问到一些具体代码的实现，比如下面两个问题：

> 题目：谈谈 jQuery 的 delegate 和 bind 有什么区别；`window.onload` 和`$().ready`有什么区别

这实际上都是 JS-Web-API 部分基础知识的实际应用：

- delegate 是事件代理（委托），bind是直接绑定事件
- onload 是浏览器部分的全部加载完成，包括页面的图片之类资源；ready 则是`DOMContentLoaded`事件，比 onload 提前一些

下面再说下比较火的 Angular、React 和 Vue。

> 为什么会出现 Angular、React 和 Vue 这种库？

理解为什么会出现一种新技术，以及新技术解决了什么问题，才能够更好地选择和运用新技术，不至于落入「喜新厌旧」的怪圈。

首先在互联网用户界面和交互越来越复杂的阶段，这些 `MV*` 库是极大提升了开发效率，比如在数据流为主的后台系统，每天打交道最多的就是数据的增删改查，这时候如果使用这些库，可以将注意力转移到数据本身来，而不再是页面交互，从而极大地提升开发效率和沟通成本。

React 还有个很好的想法是 React Native，只需要写一套代码就可以实现 Web、安卓、iOS 三端相同的效果，但是在实际使用和开发中会有比较大的坑。而且就像 Node 一样，前端用 Node 写 Server 可能需要用到的后端知识要比前端知识多，想要写好 React Native，客户端的知识也是必不可少的。React Native 和Node 都是拓展了 Web 前端工程师可以走的路，既可以向后又可以向前，所谓「全栈」。

> Angular、React 和 Vue 各自的特点

- AngularJS有着诸多特性，最为核心的是 MVVM、模块化、自动化双向数据绑定、语义化标签、依赖注入等
- React 是一个为数据提供渲染为 HTML 视图的开源 JavaScript 库，最大特点是引入 Virtual DOM，极大提升数据修改后 DOM 树的更新速度，而且也有 React Native 来做客户端开发
- Vue.js 作为后起前端框架，借鉴了 Angular 、React 等现代前端框架/库的诸多特点，并取得了相当不错的成绩。

> 一定要用这些库吗？

目前这些库的确解决了实际开发中很多问题，但是这种「三足鼎立」的状况不是最终态，会是阶段性产物。从长远来说，好的想法和点子终究会体现在语言本身特性上来，即通过这些库的想法来推动标准的改进，比如 jQuery 的很多选择器 API，最终都被 CSS3 和 HTML5 接纳和实现，也就就有了后来的 Zepto。

另外，以展现交互为主的项目**不太推荐**使用这类库，本身库的性能和体积就对页面造成极大的负担，比如笔者使用 Vue 做纯展现为主的项目，性能要比页面直出 HTML 慢。纯展现页面指的是那些以展现为主、用户交互少的页面，如文章列表页、文章详情页等。

如果是数据交互较多的页面，例如后台系统这类对性能要求不多而数据交互较多的页面，**推荐使用**。

另外，不管是什么库和框架，我们最终应该学习的是编程思维，比如分层、性能优化等，考虑视图层、组件化和工程效率问题。相信随着 ES 标准发展、摩尔定律（硬件）和浏览器的演进，目前这些问题和状况都会得到改善。

关于三者的学习资料就不补充了，因为实在是太火了，随便搜索一下就会找到。

移动开发

这里说的移动开发指的是做的项目是面向移动端的，比如 HTML5 页面、小程序等。做移动开发用的也是前面几个小节梳理的基础知识，唯一不同的是工程师面向的浏览器是移动端的浏览器或者固定的 Webview，所以会跟普通的 PC 开发有所不同。除了最基础的 JSBridge 概念之外，这里笔者重点列出以下几点：

1. 移动端更加注重性能和体验，因为移动端设备和网络都比 PC 的差一些
2. 交互跟 PC 不同，比如 touch 事件
3. 浏览器和固定的 Webview 带来了更多兼容性的问题，如微信 webview、安卓浏览器和 iOS 浏览器
4. 调试技巧更多，在 Chrome 内开发完页面，放到真机需要再调试一遍，或者需要真机配合才能实现页面的完整功能

后记

小册梳理了很多知识点，但是限于笔者精力、小册篇幅和新知识的不断涌现，难免会有考虑不到的地方，还请大家按照我在第一节提到的思维导图的方式，自己列脑图进行梳理。

最后，祝每个人都拿到满意的 offer！