Broadview®
www.broadview.com.cn

前端架构

从入门到微前端

黄峰达（Phodal）◎著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

前端架构
从入门到微前端

黄峰达（Phodal）◎著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书是一本围绕前端架构的实施手册，从基础的架构规范，到如何设计前端架构，再到采用微前端架构拆分复杂的前端应用。本书通过系统地介绍前端架构世界的方方面面，来帮助前端工程师更好地进行系统设计。

前端架构包含以下五部分内容。

- 设计：讲述了架构设计的模式，以及设计和制定前端工作流。
- 基础：通过深入构建系统、单页面应用原理、前端知识体系等，来构建出完整的前端应用架构体系。
- 实施：通过与代码结构的方式，介绍如何在企业级应用中实施组件化架构、设计系统和前后端分离架构。
- 微前端：引入 6 种微前端的概念，以及如何划分、设计微前端应用，并展示了如何实现这 6 种微前端架构。
- 演进：提出更新、迁移、重构、重写、重新架构等架构演进方式，来帮助开发人员更好地设计演进式架构

本书适合想要成为优秀前端开发工程师（初中级），或致力于构建更易于维护的系统架构的开发人员、技术主管、软件架构师和软件项目经理等。

图书在版编目（CIP）数据

前端架构：从入门到微前端 / 黄峰达著. -- 北京：电子工业出版社，2019.5
ISBN 978-7-121-36534-8

Ⅰ. ①前… Ⅱ. ①黄… Ⅲ. ①程序设计 Ⅳ. ①TP311.1

中国版本图书馆 CIP 数据核字（2019）第 092407 号

责任编辑：董 英
印 刷：三河市君旺印务有限公司
装 订：三河市君旺印务有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编：100036
开 本：787×980 1/16 印张：20.5 字数：410 千字
版 次：2019 年 5 月第 1 版
印 次：2019 年 5 月第 1 次印刷
定 价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前　言

1. 关于前端架构

独立开发一个前端项目，或身为前端技术负责人，编写代码时，总得多一分考虑。不同的场景、不同的时间、不同的角色，考虑的东西也是不同的。在多种因素的共同作用下，我们往往找不到最好的方案，只能找到最合适的方案。于是，我们需要从几个不适合的方案里选出最合适的一个，换句话说，就是从不太差的中选出最好的那个。“最合适”，便是这样相比较出来的。

过去，当人们讨论架构时，往往指的是后端架构。对于只使用后端 API 的前端来看，后端看上去像只做 CRUD（增加（Create）、读取查询（Retrieve）、更新（Update）、删除（Delete））。然而，后端并不像看上去那么简单。从架构层面考虑，后端是要实现高并发和高可用的。在多数情况下，数据库是后端最大的瓶颈，存储的时候要考虑原子性、一致性、隔离性和持久性，使用的时候要考虑通过分表、存储、主从同步来提高性能和并发量，在这个过程中还要考虑备份、迁移、查询速度、效率等问题。此外，在代码实现上还有一系列的复杂问题：使用消息队列来解耦依赖，使用微服务来拆分单体应用……

架构，在前端会更复杂——涉及领域更广。前端在实现的过程中，除了考虑代码的可用性、性能、模型构建、组件复用等问题，还有前端特有的平台设定、浏览器兼容、交互设计、用户体验等相关的问题。而在“大前端”的背景之下，还需要深入移动端设计、桌面应用、物联网等相关的领域。

后端的发展比较稳定，更注重代码实现。而前端的发展比后端晚了好多年。大概从 Google Map 在 2005 年使用了大量的 Ajax 之后，人们才意识到原来前端还可以这么做。随

后，产生了单页面应用，并诞生了一个又一个的前端应用，直至出于解耦的目的，前后端也开始分离了。

Web 前端应用越来越复杂，使得架构在前端设计与开发也越来越受关注。规范、原则、模式、架构，是我们在前端架构中需要关注的内容。

2. 本书结构

本书从结构上分为 11 个章节，除第 9 章、第 10 章存在一定的依赖性外，每一章都可以独立阅读而不受影响。

第 1 章　前端架构。介绍什么是架构、如何进行架构设计，以及相应的架构设计原则。同时，还介绍了前端架构的发展历史，以及如何通过层次设计来设计前端架构。

第 2 章　项目中的技术架构实施。介绍软件开发过程与架构的关系是怎样的，如何正确地实施架构，以及如何在过程中提升团队的能力。

第 3 章　架构基础：工作流设计。从代码的角度出发，展示前端架构的基础内容：基础规范、文档化、通过流程化提高代码质量和测试策略。

第 4 章　架构基础：设计构建流。从构建工具（webpack、Gulp、Grunt 等）入手，讲述前端应用的构建流程、如何设计构建流、持续交付的构建等，它们用于构建完整的前端应用。

第 5 章　架构设计：多页面应用。介绍如何开发传统的多页面应用，并结合前端框架的知识体系，例如模板与模板引擎、双向绑定、前端路由等，讲解如何使用合适的技术栈开发多页面应用。

第 6 章　架构设计：单页面应用。关注如何开发单页面应用，相关的内容有前端 MV* 原理、前端应用如何选型、前端应用如何启动及服务端如何渲染。

第 7 章　架构设计：组件化架构。讲述组件化架构及其三种不同的展现形式：基础的风格指南、用于重用的模式库，以及进阶的设计系统。

第 8 章　架构设计：前后端分离架构。主要关注前后端分离及其 API 管理，介绍了不同的 API 管理模式、前后端独立的模拟 API 服务 MockServer，以及服务于前端的后端（BFF）。

第 9 章　架构设计：微前端架构。包括 6 种微前端的介绍，从经典的路由分发式到多框架运行的前端微服务化等。并介绍如何划分、设计微前端应用，以及“微”害架构的相关概念。

第 10 章　微前端实战。以实战的方式，对第 9 章中介绍的 6 种微前端方案进行实践。

第 11 章　架构演进：演进式架构。介绍 5 种架构演进的方式：更新、迁移、重构、重写、重新架构，以帮助开发人员更好地设计演进式架构。

3. 本书目标

本书的目标是带领读者探索前端架构世界。我们将关注于解决有关前端应用的基础架构问题：

- 如何快速启动一个项目？
- 如何在组织内有效地共享代码、组件库、函数库等？
- 如何提升项目的代码质量？
- 项目的文档化采用什么策略？
- 采用怎样的策略进行代码测试？

……

本书还将介绍，在进行更高级的架构设计时，需要考虑的诸多因素：

- 应用的哪些部分可以在其他应用中快速重用？
- 应用内的组件采用怎样的通信机制？
- 是否通过拆分应用来降低复杂度？
- 如何对架构进行演进，以降低开发、维护成本？
- 如何让多个团队高效地进行并行开发？能否将不同应用、项目的代码隔离开来，而不是所有的人工作在一个代码库上？

……

此外，这并不是一本纯理论性书籍。书中配套了大量的相关实践代码，它们可以帮助读者更好地理解架构的实践。

4 代码

本书相关的代码都可以从 GitHub 上下载：https://github.com/phodal/aofe.code。这些代码遵循 MIT 开源协议，读者可以将这些代码用在学习、商业等用途的项目中，而不需要笔者同意。同时，笔者也不对这些代码的衍生代码负责。

5 遇到问题

在遇到问题时，欢迎随时与笔者取得联系。遇到代码问题时，建议直接在 GitHub 上创建一个相关的 issue，以便帮助其他读者解决同样的问题。

遇到内容不清楚、代码相关等问题时，可以通过下面的方式联系笔者：

（1）GitHub 上的相关项目参与讨论：https://github.com/phodal/aofe.code。

（2）知乎、微博：[@phodal]（http://weibo.com/phodal）。

（3）邮件：h@phodal.com。

（4）微信公众号：phodal-weixin。

你也可以在知乎、SegmentFault 网站上进行提问，并@phodal 来帮助你解决问题。

6. 致谢

写一本书，初期靠的是激情，中后期靠的是耐心。编程亦如此，年少的时候，笔者进入这个行业靠的是比赛，长大后靠的是好奇心。17 年过去了，笔者仍然想着不断地提升自己，因为编程领域有太多的新事物值得我们去探索。

感谢电子工业出版社博文视点的支持。

感谢女朋友@花仲马，在写作期间给予笔者的支持——笔者才能在下班之后，腾出大量的时间用于写作、编程及设计。

编码的乐趣在于创建“轮子”，在撰写本书相关内容的时候，笔者也编写了一个专用的 Markdown 编辑器 Phodit（https://www.phodit.com/）。

目　录

第 1 章　前端架构 …… 1

1.1　为什么需要软件架构 …… 2

1.1.1　什么是软件架构 …… 2
1.1.2　开发人员需要怎样的软件架构 …… 3

1.2　架构的设计 …… 4

1.2.1　收集架构需求 …… 5
1.2.2　架构模式 …… 10
1.2.3　架构设计方法 …… 11
1.2.4　生成架构产出物 …… 15

1.3　架构设计原则 …… 16

1.3.1　不多也不少 …… 16
1.3.2　演进式 …… 17
1.3.3　持续性 …… 19

1.4　前端架构发展史 …… 20

1.5　前端架构设计：层次设计 …… 21

1.5.1　系统内架构 …… 22
1.5.2　应用级架构 …… 23
1.5.3　模块级架构 …… 24
1.5.4　代码级：规范与原则 …… 25

1.6　小结 …… 25

第 2 章　项目中的技术架构实施 …… 27

2.1　技术负责人与架构 …… 28

2.2　技术准备期：探索技术架构 …… 30

2.2.1　架构设计 …… 30
2.2.2　概念验证：架构的原型证明 …… 30
2.2.3　迭代 0：搭建完整环境 …… 31
2.2.4　示例项目代码：体现规范与原则 …… 32

2.3　业务回补期：应对第一次 Deadline …… 33

2.3.1　追补业务 …… 33
2.3.2　测试：实践测试策略 …… 34
2.3.3　上线准备 …… 35
2.3.4　第一次部署：验证部署架构 …… 35
2.3.5　提升团队能力 …… 36

2.4　成长优化期：技术债务与演进 …… 39

2.4.1　偿还技术债务 …… 40
2.4.2　优化开发体验 …… 41
2.4.3　带来技术挑战 …… 41
2.4.4　架构完善及演进 …… 42

2.5　小结 …… 43

第 3 章　架构基础：工作流设计 …… 44

3.1　代码之旅：基础规范 …… 45

3.2　代码组织决定应用架构 …… 47

3.3　统一代码风格，避免架构腐烂 …… 49

3.4　使用 Lint 规范代码 …… 50

3.5　规范化命名，提升可读性 …… 51

3.5.1　命名法 …… 51
3.5.2　CSS 及其预处理器命名规则 …… 52
3.5.3　组件命名规则 …… 53

3.6　规范开发工具，提升开发效率 …… 54

3.7　项目的文档化：README 搭建指南 …… 55

3.8　绘制架构图：减少沟通成本 …… 56
3.8.1　代码生成 …… 56
3.8.2　专业工具 …… 57
3.8.3　软件附带工具 …… 57
3.8.4　在线工具 …… 58
3.9　可编辑文档库：提升协作性 …… 59
3.10　记录架构决策：轻量级架构决策记录 …… 59
3.11　可视化文档：注重代码的可读性 …… 60
3.12　看板工具：统一管理业务知识 …… 62
3.13　提交信息：每次代码提交文档化 …… 63
3.13.1　项目方式 …… 63
3.13.2　开源项目方式 …… 64
3.13.3　对比不同文档方式 …… 65
3.14　通过流程化提高代码质量 …… 66
3.14.1　代码预处理 …… 67
3.14.2　手动检视代码 …… 69
3.15　使用工具提升代码质量 …… 70
3.15.1　代码扫描工具 …… 70
3.15.2　IDE 快速重构 …… 71
3.16　测试策略 …… 72
3.16.1　单元测试 …… 73
3.16.2　组件测试 …… 75
3.16.3　契约/接口测试 …… 76
3.17　小结 …… 77

第 4 章　架构基础：设计构建流 …… 78

4.1　依赖管理工具 …… 81
4.2　软件包源管理 …… 83
4.3　前端代码的打包 …… 88
4.4　设计构建流 …… 89
4.5　持续交付问题 …… 99
4.6　小结 …… 105

第 5 章　架构设计：多页面应用 ····· 107

5.1　为什么不需要单页面应用 ····· 108
5.1.1　构建成本 ····· 108
5.1.2　学习成本 ····· 109
5.1.3　后台渲染成本 ····· 110
5.1.4　应用架构的复杂性 ····· 111

5.2　简单多页面应用的开发 ····· 112
5.2.1　选择 UI 库及框架 ····· 113
5.2.2　jQuery 和 Bootstrap 仍然好用 ····· 113
5.2.3　不使用框架：You Don't Need xxx ····· 114

5.3　复杂多页面应用的开发 ····· 115
5.3.1　模板与模板引擎原理 ····· 115
5.3.2　基于字符串的模板引擎设计 ····· 116
5.3.3　基于 JavaScript 的模板引擎设计 ····· 117
5.3.4　双向绑定原理及实践 ····· 120
5.3.5　前端路由原理及实践 ····· 124
5.3.6　两种路由类型 ····· 124
5.3.7　自造 Hash 路由管理器 ····· 125

5.4　避免散弹式架构 ····· 127
5.4.1　散弹式架构应用 ····· 127
5.4.2　如何降低散弹性架构的出现频率 ····· 128

5.5　小结 ····· 130

第 6 章　架构设计：单页面应用 ····· 131

6.1　前端 MV*原理 ····· 132

6.2　前端 MVC 架构原理 ····· 133

6.3　进阶：设计双向绑定的 MVC ····· 135

6.4　前端框架选型 ····· 138
6.4.1　选型考虑因素 ····· 139
6.4.2　框架类型：大而全还是小而美 ····· 140
6.4.3　框架：React ····· 142
6.4.4　框架：Angular ····· 143
6.4.5　框架：Vue ····· 145

6.4.6　选型总结 …… 146
6.5　启动前端应用 …… 146
6.5.1　创建应用脚手架 …… 147
6.5.2　构建组件库 …… 148
6.5.3　考虑浏览器的支持范围 …… 150
6.6　服务端渲染 …… 155
6.6.1　非 JavaScript 语言的同构渲染 …… 155
6.6.2　基于 JavaScript 语言的同构渲染 …… 157
6.6.3　预渲染 …… 158
6.7　小结 …… 159
第 7 章　架构设计：组件化架构 …… 161
7.1　前端的组件化架构 …… 161
7.2　基础：风格指南 …… 163
7.2.1　原则与模式 …… 163
7.2.2　色彩 …… 165
7.2.3　文字排印 …… 167
7.2.4　布局 …… 168
7.2.5　组件 …… 173
7.2.6　文档及其他 …… 174
7.2.7　维护风格指南 …… 174
7.3　重用：模式库 …… 175
7.3.1　组件库 …… 176
7.3.2　组件类型 …… 178
7.3.3　隔离：二次封装 …… 183
7.4　进阶：设计系统 …… 184
7.4.1　设立原则，创建模式 …… 186
7.4.2　原子设计 …… 188
7.4.3　维护与文档 …… 191
7.5　跨框架组件化 …… 192
7.5.1　框架间互相调用：Web Components …… 192
7.5.2　跨平台模式库 …… 193
7.6　小结 …… 194

第 8 章　架构设计：前后端分离架构 ····· 195

8.1　前后端分离 ····· 196
8.1.1　为什么选择前后端分离 ····· 196
8.1.2　前后端分离的开发模式 ····· 197
8.1.3　前后端分离的 API 设计 ····· 198

8.2　API 管理模式：API 文档管理方式 ····· 202

8.3　前后端并行开发：Mock Server ····· 205
8.3.1　什么是 Mock Server ····· 205
8.3.2　三种类型 Mock Server 的比较 ····· 207
8.3.3　Mock Server 的测试：契约测试 ····· 212
8.3.4　前后端并行开发总结 ····· 217

8.4　服务于前端的后端：BFF ····· 218
8.4.1　为什么使用 BFF ····· 218
8.4.2　前后端如何实现 BFF ····· 221
8.4.3　使用 GraphQL 作为 BFF ····· 223

8.5　小结 ····· 228

第 9 章　架构设计：微前端架构 ····· 229

9.1　微前端 ····· 230
9.1.1　微前端架构 ····· 230
9.1.2　为什么需要微前端 ····· 232

9.2　微前端的技术拆分方式 ····· 234
9.2.1　路由分发式 ····· 235
9.2.2　前端微服务化 ····· 236
9.2.3　组合式集成：微应用化 ····· 237
9.2.4　微件化 ····· 238
9.2.5　前端容器：iframe ····· 239
9.2.6　结合 Web Components 构建 ····· 240

9.3　微前端的业务划分方式 ····· 241
9.3.1　按照业务拆分 ····· 242
9.3.2　按照权限拆分 ····· 243
9.3.3　按照变更的频率拆分 ····· 243
9.3.4　按照组织结构拆分 ····· 244

9.3.5　跟随后端微服务拆分……244
9.3.6　DDD 与事件风暴……245
9.4　微前端的架构设计……245
9.4.1　构建基础设施……246
9.4.2　提取组件与模式库……246
9.4.3　应用通信机制……247
9.4.4　数据管理……248
9.4.5　专用的构建系统……249
9.5　微前端的架构模式……249
9.5.1　基座模式……250
9.5.2　自组织模式……251
9.6　微前端的设计理念……252
9.6.1　中心化：应用注册表……252
9.6.2　标识化应用……253
9.6.3　生命周期……253
9.6.4　高内聚，低耦合……254
9.7　“微”害架构……254
9.7.1　微架构……256
9.7.2　架构的演进……256
9.7.3　微架构带来的问题……257
9.7.4　解决方式：可拆分式微架构……259
9.8　小结……259
第 10 章　微前端实战……261
10.1　遗留系统：路由分发……262
10.1.1　路由分发式微前端……263
10.1.2　路由分发的测试……264
10.2　遗留系统微前端：使用 iframe 作为容器……266
10.3　微应用化……266
10.3.1　微应用化……267
10.3.2　架构实施……269
10.3.3　测试策略……271
10.4　前端微服务化……272

10.4.1 微服务化设计方案 ····· 273
10.4.2 通用型前端微服务化：Single-SPA ····· 276
10.4.3 定制型前端微服务化：Mooa ····· 279
10.4.4 前端微服务化总结 ····· 283
10.5 组件化微前端：微件化 ····· 283
10.5.1 运行时编译微件化：动态组件渲染 ····· 284
10.5.2 预编译微件化 ····· 287
10.6 面向未来：Web Components ····· 288
10.6.1 Web Components ····· 289
10.6.2 纯 Web Components 方式 ····· 291
10.6.3 结合 Web Components 方式 ····· 293
10.7 小结 ····· 295
第 11 章 架构演进：演进式架构 ····· 297
11.1 更新 ····· 298
11.1.1 依赖和框架版本升级 ····· 299
11.1.2 语言版本升级 ····· 300
11.1.3 遗留系统重搭 ····· 300
11.2 迁移 ····· 301
11.2.1 架构迁移的模式 ····· 302
11.2.2 迁移方式：微前端 ····· 303
11.2.3 迁移方式：寻找容器 ····· 303
11.3 重构 ····· 304
11.3.1 架构重构 ····· 304
11.3.2 组件提取、函数提取、样式提取 ····· 305
11.3.3 引入新技术 ····· 306
11.4 重写 ····· 307
11.4.1 重写能解决问题吗 ····· 308
11.4.2 梳理业务 ····· 309
11.4.3 沉淀新架构 ····· 310
11.5 重新架构 ····· 311
11.5.1 重搭架构 ····· 311
11.5.2 增量改写 ····· 312
11.6 小结 ····· 313

1

第 1 章
前端架构

没有一种架构能满足未来的需求。

在软件的生命周期中，架构可以不断地优化，持续地变好，使得架构可以适用于当前的场景。所以架构是可以改变的，架构是需要变化的。

架构往往都是在一定的约束条件下设计出来的。在设计的时候，架构师总会担心设计出来的架构不能满足未来的需求，容易遭到后来人的吐槽。可受限于当前的时间、人力、财力、环境、能力，我们设计出来的架构，只是符合当前约束的架构。既然如此，我们就要以一种开放的心态来看待这个问题。未来的接任者，在经历了一定的练习之后，能带领团队演进出更好的架构。

架构也是分层级的，在不同的阶段里形式是不一样的，当面向不同的人群时，模式也是不一样的。所以每个人所理解的架构就会有所差异，那么到底什么是架构呢？

1.1 为什么需要软件架构

1.1.1 什么是软件架构

对于软件架构，不同的人有不同的理解，不同的组织有不同的定义。如维基百科上说：

软件架构是指软件系统的高级结构，以及创建这种结构和系统的约束。每个结构包括软件元素、元素之间的关系，以及元素和关系的属性。软件系统的架构是一种隐喻，类似于建筑物的体系结构。它作为系统和开发项目的蓝图，列出了设计团队必须执行的任务。

而在 IEEE（1471 2000）中，架构是指体现在它的组件中的一个系统的基本组织、组织之间的关系、组织与环境的关系及指导其设计和发展的原则。

又或者相似的其他定义，无一不在强调设计架构的整体及内部各个部分的关系。与此同时，还涉及实施过程中的原则和相关的设计实践。软件开发的过程就好比盖房子：

◎ 如果不能完成地基的建设，那么房子就无法继续往上盖。

◎ 开始盖房子时，便是在实施对应的架构。若地基有问题，房子就难以长期存在。

◎ 在浇筑钢筋水泥的过程中，若不注意，房屋就可能歪歪扭扭。

◎ 虽然房子盖好了，但是不注意后期的保养和维护，仍然会出现问题。

盖房子和软件开发是相似的，需要规划、设计、实施，而后才能使用。盖房子，最先设计的也是建筑的架构，有了建筑的蓝图，还需要考虑美观、实用、坚固等要素，从各种各样的材料中选择合适的，完成这一系列决策才能进入实施阶段。在实施的时候，还需要严格按照建筑的蓝图来进行施工，并保证施工的质量，才能造出所需的建筑。

对于软件开发来说也是相似的。最初，设计出软件的总体架构蓝图，思考各个模块之间的关系，实施一系列相关的架构决策。然后，选择软件开发所需要的一系列技术栈、框架等，讨论关于应用的上线、部署等流程问题。最后，才能进入软件的开发阶段。在开发的过程中，还需要保证软件的质量，才能设计出符合要求的系统。