全栈应用开发：精益实践

作者：黄峰达

内容简介

这不是一本深入前端、后台、运维、设计、分析等各个领域的书籍。本书以实践的方式，将这一系列的领域及理论知识结合到一起，来帮助读者构建全栈Web开发的知识体系，并辅以精益及敏捷的思想，来一步步开发Web应用：从创建一个UI原型到编写出静态的前端页面；从静态的前端页面到带后台的应用，并部署应用；从Web后台开发API到开发移动Web应用。

在这个过程中，我们还将介绍一些相辅相成的步骤：使用构建系统来加速Web应用的开发；为应用添加数据分析工具来改进产品；使用分析工具来改善应用的性能；通过自动化部署来加快上线流程；从而帮助读者开发出一个真正可用的全栈Web应用。同时，我们也将帮助读者把这些步骤应用到现有的系统上，改进现有系统的开发流程。

图书在版编目（CIP）数据

全栈应用开发：精益实践 / 黄峰达著. —北京：电子工业出版社，2017.5

ISBN 978-7-121-31369-1

I. ①全… II. ①黄… III. ①网页制作工具－程序设计 IV. ①TP393.092.2

中国版本图书馆CIP数据核字（2017）第078297号

策划编辑：董 英

责任编辑：李利健

印 刷：

装 订：

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：24.5 字数：441.2千字

版 次：2017年5月第1版

印 次：2017年5月第1次印刷

印 数：3000册 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

学习Web开发最难的不是学习相关技术，而是需要了解整个Web开发的知识体系。多数时候并不是因为我们不学习，而是因为我们不知道学习什么。完整的知识体系不仅仅包括前端、后台开发，还应该包括持续集成、自动化部署等内容。这些往往需要几本不同的书才能学习到，另外，它们也难以保证知识体系的完整性。我们在学习的时候，也往往并没有注意到它们之间的联系。

本书可以为读者构建出清晰、完整的Web开发体系，包括：前端、后台的技术选型，搭建构建系统，如何上线部署，并进行数据分析，以及如何在其中结合最好的工程实践等。

希望作为读者的你，可以将本书当作一本索引书籍，以此来开启你的Web开发新世界；你可以按书中的实践来进行Web编程，并结合理论来实践。

为什么写这本书

本书是我在实习的时候特别想写的一些内容——关于如何系统地学习Web开发，只是我一直缺少一条主线来将这些内容一一串起来。

2016年年初，我在GitHub上开源了一个名为Growth的应用（读者可以在App Store和各大应用商店下载该软件）。在该应用中便包含了本书的主要思想：Web应用的生命周期。在不断迭代的过程中，该应用越来越受开发者喜爱，至今已经有超过10000名用户用过这个应用。随后，笔者在GitHub上推出了开源电子书《Growth：全栈增长工程师指南》，已经有超过4500个Star。由于电子书本身只是一个指南，越来越多的读者还希望有一本实战。也因此诞生了《Growth：全栈增长工程师实战》，其在GitHub上也有超过1000个Star。

后来，我才下决心去出版这样一本书。写一本书不是一件容易的事，相比较而言，读一本书则要简单许多。前者要花费一个人几个月的时间来完成，而后者只需要几星期、几天，或者是几小时的事。花几分钟将书的目录过一遍，随后只看几页想看的内容，余下的内容则可以在以后闲暇的日子里探索。

本书是我在编程生涯初期的一些体会，它更像是一本关于Web开发的索引书籍，但其实这些索引正是我读了大量书籍后，自己对精髓之处进行的理解加工。在这本书里，你会看到我对很多知识点进行了概括，并以实践的方式将一个个知识点连接到一起。

在最开始的时候，我曾经想将书名命名为“实习记”。后来又觉得虽然这是在我实习期间学到的知识，但其实很多内容在其他公司是学不到的。因此，在电子书里将其命名为Growth，它不仅可以使读者增长知识，也在让我自己成长。

本书目标

本书的目标是帮助读者构建Web应用的全栈开发所需要的完整知识体系，并以精益创业的思想来一步步开发Web应用。

- 从创建一个UI原型到编写出静态的前端页面。
- 从静态的前端页面到后台的应用，并部署应用。
- 从Web后台开发API到开发移动Web应用。

在这个过程中，我们还将介绍一些相辅相成的步骤：

- 使用构建系统来加速Web应用的开发。
- 为应用数据分析工具改进产品。
- 使用分析工具改善应用的性能。
- 通过自动化部署加快上线流程。

从而帮助读者开发出一个真正可用的全栈Web应用。同时，我们也希望能帮助读者将这些步骤应用到现有的系统上，改进现有系统的开发流程。

本书结构

本书从结构上分成了3部分，每部分都会有不同的侧重点。

第1部分：准备阶段

在这一部分里，我们将主要集中于编码前的一系列开发准备工作，从选择一个合适的IDE到创建一个Web应用的构建流。

第1章　基础知识　介绍了搭建开发所需要的基本环境，以及IDE、操作系统、版本管理工具等日常工具的选择与使用；还介绍了如何对一个目标进行切分，以便我们在实现的时候可以一步步往下实践。

第2章　最小可行化应用　介绍了如何使用UI工具来创建原型，并根据这个原型创建出一个最简单的Web应用；接着介绍了在Web应用开发的过程中，如何使用精益的思想来开发出用户喜爱的产品。

第3章　技术选型与业务　对后台开发所需要的技术进行简单概览，并介绍了不同后台组件的框架，以及如何从这些框架中选择出合适的框架。同时还介绍了Python下的Web开发框架Django，以及如何用这个框架创建一个“hello，world！”程序。

第4章　构建系统及其工作流　介绍Web应用中常见的构建流程及组件，以及如何结合Fabric打造后台的构建系统。

第2部分：编码到上线

在这一部分里，我们主要讲述大部分Web应用的开发过程，并介绍在开发过程中一些好的实践。

第5章　编码　介绍了如何使用Django创建一个简单的博客应用，以及如何使用单元测试、UI测试来测试代码的功能。

第6章　上线　介绍了如何手动部署开发的Web应用到产品环境，以及如何使用自动部署工具来完成自动化部署。

第7章　数据分析和体验优化　介绍了如何使用网页监测工具来分析网页的流量来源、用户行为等，并结合一些前端、后台的优化工具对应用进行优化。

第8章　持续集成与持续交付　介绍了如何使用持续集成工具，以及如何使用持续集成工具来改进开发流程，并实现自动化的部署。

第9章　移动Web与混合应用　介绍如何结写后台API来创建移动应用，以及如何为单页面应用提供SEO支持。

第3部分：增量性优化

第10章　遗留代码与重构　介绍什么是遗留系统，以及如何基于第2部分中的经验来改进遗留系统。

第11章　增长与新架构　介绍如何使用回顾与反馈来使程序员成长，以及如何依据需要设计出新的架构。

技术栈概述

本书所介绍的工具主要集中于前端、后台、构建工具和前端UI框架四部分，分别如下。

- Django是Python语言的一个MVC架构Web开发框架。本书使用这个框架来介绍如何编写单元测试、功能测试，并演示如何使用它进行持续集成和持续部署。

- Bootstrap是一个在前端领域相当流行的响应式Web UI开发框架，本书出于开发便捷的缘故使用这个框架。
- Fabric是一个命令行的自动化部署工具，本书使用这个框架来展示如何搭建构建系统，并使用它来进行自动化部署。
- Angular 2是一个可以用于构建移动应用和桌面Web应用的开发平台，我们在书里用它来展示如何开发前后端分离的Web应用程序。

上面的几个框架可以构成跨手机、桌面的一个Web应用，以及如何对其进行自动化部署。另外，还将介绍一些工具和框架来帮助我们开发：

- Ionic 2是一个跨平台（Android、iOS、Windows Phone）的混合应用开发框架，基于Angular 2框架，并搭建有大量的UI组件，以及原生组件，我们在书里说明如何通过它与Angular 2共用代码来开发手机端应用。
- Jenkins是一个持续集成工具，它提供了持续集成与持续部署工具链中所需要的大部分工具。我们将用它来展示如何进行持续集成，并结合Fabric来实现自动化部署。

本书将展示如何结合这些工具来做一些最佳实践，读者不必担心它会影响你的阅读，并且这些工具的替代品也很容易找到。

代码

本书相关的代码都可以从GitHub上下载到：https://github.com/phodal/growth-code。

混合应用部分的代码可以从https://github.com/phodal/growth-paper-hybrid处下载。

这些代码遵循MIT协议开源，读者可以将这些代码用于学习、商业等用途的项目中，不需要笔者授权。同时，笔者也不对这些代码的衍生代码负责。

遇到问题

在遇到问题时，欢迎及时与笔者联系。遇到代码问题时，建议直接在GitHub上创建一个相关Issue，以便我们帮助其他读者解决同样的问题。

遇到内容不清楚等问题时，可以通过下面的方式联系笔者：

1．通过GitHub上的Growth项目参与讨论：https://github.com/phodal/growth-code

2．在Growth论坛上讨论：https://forum.growth.ren/

3．在微博上与我联系：@phodal

4．通过邮件：h@phodal.com

5．加入QQ群讨论：529600394

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

致谢

我要把这本书献给花仲马，没有她，就没有这本书。感谢她在这本书的写作过程中一直陪伴着我，并为这本书进行了中文校对来保证语句的通顺。

同时，我想特别感谢ThoughtWorks的同事薛倩、阿里巴巴的孙辉在本书创作过程中提供了详细的反馈，正是他们的帮助让本书更加准确、容易阅读。我还想特别感谢在ThoughtWorks学习时的同事，为我提供悉心指导与帮助。特别感谢王超、陈卿、王妮、曹隆凯、张静强、刘杰、王磊，在和他们进行结对编程时，我学习到了敏捷软件开发、Tasking等编程之外的技能，感谢他们帮我走了这么远。

此外，还有那些在GitHub上为我提供反馈的用户，正是他们的反馈促使这本书更加完整。由于人数众多，这里仅列出这些用户的ID：

感谢ethan-funny、izhangzhihao、kaiguo、gymgle、aidewoode、wenzhixin、sasuke6、wangyufeng0615、walterlv、lolosssss、NehzUx、mikulely、yulongjun、PhilipTang、ReadmeCritic、ReadmeCritic、wangcongyi、loveisbug等用户为《Growth：全栈增长工程师指南》提供反馈与修改。

感谢Pandoraemon、wo0d、ReadmeCritic、zhangmx、felixglow等用户为《Growth：全栈增长工程师实战》提供了反馈与修改。

目录

第0章　绪论：Web应用开发周期

这部分内容最早出自笔者写的文章《RePractise：Web开发的七天里》，原文简单描述了Web应用的生命周期。后来发现，这条路几乎是所有Web应用的必经之路。一个Web应用在其生命周期里，都要经历搭建开发环境、创建构建系统、编写代码、进行数据分析等，直至最后使用新的系统来替换这个遗留系统。

作为本书的开头，笔者难免想说些废话，初学者可以跳过这一部分。等到阅读完本书再看看这部分内容，或者等完全经历了一个项目的开发过程，再回过头来看这部分的内容就会有所体会。如果你是一个有经验的开发者，相信你对这个生命周期一定也深有体会。

0.1 Web应用的生命周期

在我所经历的项目以及我所看到的Web应用里，它们都有相同的很有意思的生命周期。我们经常在网上看到某个知名的网站使用某个新的技术、语言来替换旧的系统，某个App使用新的框架来替换现有的App。我们所看到的都只是这些公司正在重构现有的系统，这实际上是一个周期的结束，以及一个新周期开始。其过程如图0-1所示。

图0-1 Web应用的生命周期

仔细一想就会发现：我们所经历的项目都在以不同的时间长度经历相同的生命周期。

0.2 遗留系统与新架构

在我开始工作的时候，接触的第一个项目就是一个遗留系统。在一次休息时，我们在比赛找最古老的源码文件，最后找到了10年前写下的一个文件。尽管在我们的代码里有单元测试、针对具体业务功能的测试，项目的代码已经超过20万行，项目中仍然有相当多的代码超出了我们所理解的业务范围。毕竟在这些年头里，有相当多的功能已经不存在了。后来，我们选用微服务重构了这个系统。对于中型的遗留系统来说，这算是一剂良药。

让我们先从某某网站使用新架构重新设计说起。当我们决定使用新架构重新设计系统时，原因可能是多种多样的，如果我们排除一些无法抗拒的因素（如政治），那么剩下的原因可能就只有两个。

- 系统已经变得难以维护。这里的原因仍然有很多：大量的代码已经没有人知道其业务逻辑，变得难以修改；代码间耦合度过高，重构系统的难度过于复杂；项目所使用的技术栈已经过时，已经被市场所淘汰；团队的技术栈在成员变动的过程中，团队中大部分成员的技术栈已经和当前的项目不匹配了。
- 系统的技术栈已经难以符合业务的需求。绝大多数情况下，我们在最初开始创建项目的时候，所选择的技术栈都是符合当时业务需求的技术栈、可以快速验证其业务价值的技术栈。而随着业务的扩张，现有的技术栈很快将难以满足当前业务的需求，或出现性能优化上的限制。

在多数情况下，我们都会将这种系统称为遗留系统。在这时团队里的气氛便是“能不动这些代码就尽量不去动它”。我们已经很难将项目的问题归为人的因素，多数时候都是受业务扩张的影响。作为一个专业的程序员，我们的本能就是将程序写好，而我们往往没有这样的机会。

业务人员对项目经理说：“我们的竞争对手已经在本周上线了这个功能。”

项目经理对开发人员说：“这个功能下星期就要上线！”

是的，我们的功能不得不马上上线。这时候，我们往往要在代码质量和交付速度上做出一些妥协。妥协多了，系统也就变烂了。

开发人员说：“这个代码我不太敢修改，要是出了什么大Bug怎么办？”慢慢地人们就开始讨论起重构系统的事宜，并开始着手设计新的架构——使之可以满足当前的业务需求、可预测时间内的业务与技术需求。

0.3 技术选型与验证

在讨论新架构的过程中，不同的人可能会有不同的技术偏好，也会因存在一些政治因素导致不同技术方案的产生。如团队中的一些人可能出于稳定缘故而选择Java，一些人可能出于对新技术的需求选择Scala，而另外一些人可能考虑到团队中大部分人可能因为都会使用JavaScript而选择使用JavaScript。如图0-2所示，我们的考虑应该不仅仅取决于这一系列的技术因素。

图0-2 技术选型考虑因素

需要注意的是：在做技术选型的时候，要尽最大可能以团队为核心。在做决定之前，我们要提出不同语言、框架下的技术模型，并且进行验证。随后就需要快速搭建出一个原型，并针对这个原型进行假想式开发，然后验证原型本身是经得起考验的。

在这一阶段，我通常喜欢在GitHub上搜索一些名字中带有boilerplate的项目，即模块文件。而当一个框架很流行的时候，我就会去相应的awesome-xx寻找，如awesome-react就可以寻找到react相关的项目集。然后，克隆这样一个项目，开始依照现有的系统创建简单的Demo。随后，就可以依据我们的业务试着在这上面进行扩展。最后，再决定是否使用这门技术和这个框架。

通常来说，在选择一门新技术设计系统时，需要承担的风险相当大，而如果能成功，那么它可能会带来巨大的收益。从这点看，使用最新的技术与赌博无异。在一些成熟的公司里，会有专门的技术委员会负责对新技术进行审核，来决定是否可以在某个项目里使用新技术。除了考虑其为开发带来的便利性，他们更多地还会考虑其成熟度、安全和技术风险等。

0.4 搭建构建系统

决定好架构并选择完技术栈后，我们就开始着手创建项目的构建系统，设计项目的部署流程。构建系统不仅包含项目相关的构建流程，还从某种意义上反映了这个项目的工作流程。

创建完“hello，world”程序后，我们要着手做的事情就是创建一个持续集成环境。这样的环境包含一系列的工具、步骤及实践，从工具上说，我们需要选择版本管理工具、代码托管环境、持续集成工具、打包工具、自动部署脚本等一系列流程，这些流程将会在第4章详细讨论。

图0-3便是笔者之前经历过的一个项目的构建流程。

图0-3 构建过程

这是一个后台语言用Java、前台语言用JavaScript的项目的构建流程。

0.5 迭代

在互联网行业里，能越快速地对市场需求做出反应，就越能有更好的发展。只要你细心观察就可以发现，大部分互联网公司都在以一定的规律更新产品，或者一周，或者两周，又或者一个月等，这种不断根据反馈来改进产品的过程称为迭代。如图0-4所示是一个简化的迭代模型。

图0-4 简化的迭代模型

当一个迭代开始时，我们需要收集上一个迭代的反馈或者新的需求，然后开始开发代码，最后再发布产品。开发的产品在这个过程中不断地增强功能。为此，还需要选择一个好的迭代周期。一个好的迭代周期既应该有充足的时间修复上一个迭代的Bug，又能在下一个迭代开始之前交付重要的功能。当然，如果交付的软件包里出现了重要的Bug，那么我们也能在第一时间使用旧版本的包，并在下一个迭代交付。在这样的开发节奏里，一周显得太短，一个月又显得太长，两周会是一个很不错的时间。

当一个团队在这方面做得不好时，那么他们可能在一次上线后，发现重要的Bug，不得不在当晚或者第二天更新他们的产品。即使是有经验的团队，在开发初期也会经常遇到这些问题，而这些问题可以依赖于在迭代中改进。好的迭代实践都是依据团队自身的需求而发展的，这意味着有时候适合团队A的实践并不一定适合团队B。

随后，我们会在这个“hello，world”的基础上不断添加各种功能。

0.6 Web应用开发步骤

令人难以置信的是，我们做了这么多事情以后还没有开始写代码。事实上，在这一步里，我们已经搭建好了一个最小可运行的Web应用。在这之后，我们所要做的事情就是提交代码即可。将代码从本地提交到服务器后，持续集成服务器将帮我们运行测试，在测试通过后，打包、发布现有的代码，最后部署到测试环境里。

（1）编码

如果不考虑技术难度的因素，写代码看上去就是一件很简单的事。我们只需要按照需求，将功能一点点往上叠加即可。如果不考虑这个过程中添加的代码质量，将会得到一个难以维护的系统，并且在拿到需求后的第一反应也并非直接开始实现功能，而是首先应该考虑可以将这个需求拆分为几步，我们将这个过程称为Tasking。

假如，我们正在实现某个详情页的显示功能，它依赖于前端和后台。那么可以直接先做后台API，再实现前台API，最后依据需要微微调整API。我们也可以先用Mock的数据实现前端页面，再依据定义出来的数据格式实现后台API。在这两种不同的实现中，我们都有一个明确的先后步骤。同样，对于一个更加复杂的功能来说，需要切分得更加仔细，每一次只挑选其中一个任务，实现后，再一步步往下执行，最后实现这个功能。

有意思的是，当我们已经决定切分为多步来实现功能的时候，就可以在每一步里进行几次不同的代码提交，以便以后知道每一步中做了什么内容。如果只是在最后一步直接提交代码，那么在未来修改代码时，便难以理清当时的思路。

一个合理的编码过程不仅包括功能的实现，还应该有测试。尽管出于项目进度的原因，多数项目都不存在测试，而正是因为没有测试，使得整个项目更加混乱。新的功能容易影响旧有的代码，除非有足够多的测试人员，否则我们无法保证所有的功能都是正常的。在有限的条件下，我们应该编写重要的测试，以保证核心功能不被破坏。在条件允许的情况下，我们应该尽可能地保证测试对重要功能的覆盖。由于代码库不只有一个人在提交，如果在某次提交中测试被破坏了，就可以知道谁破坏了测试，他/她应该有责任来修复这个测试。

在完成功能后，我们还可以对代码进行重构，以此来保证代码的质量。此外，在日常工作中，我们会用Code Diff（代码检视）来帮助大家提高代码质量。因此，并不是实现了功能就完事了，我们应该尽量保证代码的质量。

（2）上线和数据分析

好了，现在是时候上线了。在以前，上线就是登录到服务器做数据备份。随后，在本地构建、上传软件包，安装软件的依赖。最后，重启服务器、Done。

在今天看来，这是一件相当费力的事，我们可以使用自动部署工具来加快这个流程，甚至当我们有足够的测试覆盖率时，可以直接将测试通过的代码直接部署到产品环境。不过，要这样做应有相当的技术能力，并且要保证我们可以协调好开发人员、运维人员等。从技术上说，这可能是一件容易的事，但是从组织结构上说，这并不是一件轻松的事。

而故事并没有因此而止步于上线，在产品上线时，我们可以通过数据分析工具来监测用户的行为、网站的访问量等信息。

对开发人员来说，这样的分析平台可以帮助我们解决用户在使用过程中遇到的Bug——他在哪一步出的问题？他在出问题前做了什么操作？

对业务人员来说，他们可以借此来分析产品受欢迎的程度、用户及流量来源、转化率等信息，并依此来对着陆页、转化率等进行优化。几种常见的流量来源包括搜索引擎、外部链接、付费搜索等，这些都可以依此来做出一些调整。从技术角度说，我们可以提高网站的SEO（搜索引擎优化）水平来添加流量，这将在第7章中进行讨论。

0.7 小结

本章我们对本书的内容进行了一个简单概述，并完整地介绍了它们之间的联系。同时还介绍了在阅读过程中，我们将学习到的内容，以及将遇到的一些挑战。

第1部分　准备阶段

在这一部分里，我们将主要精力集中于“项目开始前”的一些准备工作，如搭建基础的构建系统、从业务角度对技术进行选型等。同时，我们还将关注一些特别有意思的东西，如Web应用的生命周期、对不同业务的技术栈考虑等。