iOS
企业级应用开发技术

和凌志 / 著

简述了如何开发一个可上线、可运营的App，详细介绍了原生与HTML5混合开发模式。

适合想熟悉iOS企业级App开发的，以及想学习前端（Web）和全栈技术的工程师。

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

iOS
企业级应用开发技术

和凌志 / 著

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

内 容 简 介

本书聚焦在 App“产品”的设计、开发和运营层面，特别强调架构和设计模式的重要性，有意识地将设计模式应用到代码的编写中，重点介绍 iOS 企业级应用开发的设计思维方式，并与全栈开发技术结合起来。

全书分为 iOS 基础篇、Web 与 Native 混合开发模式篇和全栈开发技术篇。iOS 基础篇主要介绍 iOS 基础知识、多种设计模式下的视图控制器之间的传值、App 与服务器接口的定义、CollectionView 的应用；Web 与 Native 混合开发模式篇主要介绍 Block 的应用、iOS 网络请求、JavaScript 基础、Web 与 Native 的交互；全栈开发技术篇主要介绍 Node.js、Express、AngularJS、MongoDB、MEAN 全栈技术的实现。

本书适合从事 iOS 开发的人员，以及有一定 iOS 基础、想学习全栈技术的人员阅读。

源码下载地址：https://github.com/leopard168/iOS-Enterprise。

图书在版编目（CIP）数据

iOS 企业级应用开发技术 / 和凌志著. — 北京：电子工业出版社，2017.10
ISBN 978-7-121-32828-2

I. ①i… II. ①和… III. ①移动终端－应用程序－程序设计 IV. ①TN929.53

中国版本图书馆 CIP 数据核字(2017)第 244091 号

责任编辑：田宏峰
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：16.25　字数：366 千字
版　　次：2017 年 10 月第 1 版
印　　次：2017 年 10 月第 1 次印刷
定　　价：68.00 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：tianhf@phei.com.cn。

序

认识和凌志先生，是在 2013 年。

在那一年，北方工业大学计算机学院新增了新的数字媒体技术实验室，作为学生创新、创业、企业实训培养基地，为学生提供了更好的设计与开发实验环境。为了进一步激发学生的学习兴趣，促进产学研的结合，同时紧跟技术发展趋势，我们开设了一门基于移动终端的互联网课程，聘请经验丰富的和凌志先生讲授“iOS 企业级应用开发技术”。

和凌志早期担任西门子手机软件平台架构师，近十多年来，一直从事移动互联网的产品开发工作，在电商平台、移动流媒体方面，积累了丰富的行业经验。他把这些宝贵的企业经验应用在了教学实践中。早在开课之前，和凌志就已经完成了《iOS 开发之美》一书，该书通俗易懂，很适合初学者快速上手。

为了给学生创造一个更好的学习条件，我们与北京高校计算机信息类专业群合作，特意录制了 iOS 开发的 MOOC（慕课）视频课程，从 Objective-C 编程语言、表视图，到网络请求、数据存储，都给出了详细的实例。

与市面上已有的 iOS 参考书不同的是，《iOS 企业级应用开发技术》一书聚焦在“产品”的设计、开发和运营层面。所谓企业级应用，指开发出来的产品是可上线、可运营的，而不是仅仅用来演示的，而且企业级应用还要考虑开发周期、研发成本、运行和维护的投入。

书中特别强调架构和设计模式的重要性，坚持倡导一个理念：“在一个给定的场景下，最佳方案只有一个”。比如，在讲述视图控制器之间的传值时，先后列举了五种不同的实现方案，对比不同的方案，给出相应的应用场景；在讲述 Web 与 Native 交互时，给出了多个应用实例；类似的场景还有很多。

从书名上看，《iOS 企业级应用开发技术》的主题是在讲述 iOS 开发技术，但作者并没有仅仅停留在 iOS 开发本身，而是延伸到“全栈（Full Stack）”技术路线上。以 Node.js 为平台的全栈技术越来越受到互联网公司的热捧。

当下，我们正处于共享经济的时代，知识分享也日益成为一种时尚。通过尝试校企合作，

从适应技术发展的角度提升学校课程设置使之与社会需求紧密结合，为学生在校期间掌握前沿技术提供一种良好手段，为毕业后走向社会打下扎实的技术基础。

马 礼

北方工业大学计算机学院院长

前　言

缘起

为何要写一本《iOS 企业级应用开发技术》的书？这还得从我的工作经历说起。

早在 2014 年，北方工业大学计算机学院就创建了数字媒体技术实验室，并为此配备了国内一流的实验设备。在过去的几年，我一直为学生们讲授“iOS 企业级应用开发”课程，本书记录了学生实战的心路历程。

本书成稿历经两年，其间几易其稿。在这个过程中，北方工业大学计算机学院的学生和老师，都给我了很多无私的帮助。每次审视这些教学案例，都会得到一次技术上的升华。在本书出版之前，初稿曾在北方工业大学计算机学院试用，在整个教学过程中，我为学生们在课堂上勇于“质问”的精神点赞。

北方工业大学计算机学院数字媒体技术实验室

读者对象

本书所面向的读者对象是有一定 iOS 基础的开发者。如果你是第一次接触 iOS 开发，建议还是翻阅一下笔者的拙作——《iOS 开发之美》。

既然书名命名为“iOS 企业级应用开发技术”，说明它不是一本泛泛的 iOS 基础的普及，相反，本书所关注的是，在企业项目开发中碰到疑难杂症时，如何开具良方，对症下药。

具体来说，这本书适宜的读者有：

（1）想学习 Objective-C 2.0 编程语言的初学者。

过去的几年，几乎每年的 WWDC 开发者大会，苹果公司都会隆重地推广 Swift 语言，但现实情况是，在 iOS 企业级应用开发中，Objective-C 依然是主流。

作为一门编程语言，Swift 还在处于不断的更新中，从早期的 1.0，发展到今天的 3.0，可见其变化之大。从另一个角度来讲，说明它还存在不稳定性。

本书的所有实例，都是基于 Objective-C 2.0 编写的。尽管 Objective-C 与 Swift 都是基于 Xcode 进行开发的，从设计模式上讲，它们有异曲同工之妙；但从编程风格上讲，二者的差异还是蛮大的。

（2）有了一定的 iOS 基础，想扩展 iOS 知识面的。

就 iOS 技能而言，本书花了大量篇幅介绍：不同设计模式下的视图控制器之间的传值；对表视图（UITableView）一笔带过，而对集合视图（UICollectionView）则通过多个维度来演练，这是因为在笔者看来，表视图是 iOS 的基础控件，而在其之上，才是集合视图；还有 Block（块）——这个令初学者感到望而生畏的地方；网络请求、断点下载，都在本书中娓娓道来；在构建企业级应用时，混合开发模式越来越受到青睐，书中详细讲述了 Native 与 Web 的交互。

（3）已经在 iOS 领域久经沙场，想学习“大前端”、“全端”及“全栈”的。

随着 App 多年的发展，App 的优势和短板日益明显，原生技术无法解决的问题，需要前端技术（HTML5）来弥补，二者才能相得益彰，所以混合开发模式越来越受欢迎。

常听到有人感叹，iOS App 的用武之地没有想象得那么广阔，即使不转型，也想再扩展一下自己的技术路线。从产品形态上看，iOS App 也属于前端范畴，这里，笔者给出的建议是学点前端技术，掌握点 JavaScript，向“全栈”进军。

如果一个 App 开发工程师同时具备了原生与全端的技能，由“单翼”变成了“双翼”，其技术路线的前景将变得越来越广阔。

如何阅读本书

作为一门 iOS 企业级应用开发技术，其蕴含的知识点无疑有多个方面。本书的特点是，针对同一个应用场景，用不同的方案来实现它，经过演绎推理，得出一个结论——在给定的场景下，最佳方案只有一个！

本书分为 iOS 基础篇、Web 与 Native 混合开发模式、全栈开发技术。

（1）iOS 基础篇。

就 iOS 技能而言，本书先介绍了 Objective-C 语法，再过渡到定制化视图的创建和管理，以及 App 与服务器接口的定义。此外，本篇还有两个重要部分。

视图控制器之间的传值：总结以往的项目经验，我们发现，视图控制器之间的传值是必不可少的，而且实现的方式有多种。在同一个项目中，对同一类问题的解决会出现多个“门派”，如果不统一约定，后期维护将非常被动。为便于理解，这里给出了同一个场景下的 4 种实现方案（Delegate、Singleton、KVO、Notification），还有一种 Block 实现方式，放在了混合开发模式中讲解。以上实现方案，每一种方法的背后都有一种设计模式的支撑。只有掌握了设计模式，才能更好地理解和融会贯通。

集合视图的应用：我们没有讲述传统的表视图，而是聚焦在集合视图（UICollectionView）上，旨在说明，如要构建一个华丽的 UI 页面，就要善用集合视图。通过自定义的 UICollectionViewLayout，可以轻松地实现一个瀑布流，而这种瀑布流效果是表视图所无法比拟的。为了能够掌握集合视图的应用，书中给出了大量的实例。

iOS 的基础知识是没有穷尽的，只要掌握了核心的实现模式，其他的便迎刃而解，正所谓“一叶知秋”！

（2）Web 与 Native 混合开发模式。

当我们面向企业级 App 开发时，需要用一种产品设计的思路来引导学习技术路线，而不是仅仅停留在 iOS 知识本身。之所以采用 Web 与 Native 混合开发模式，是因为 iOS 原生开发有着明显的短板，而 Web（HTML5）为产品的推广带来了新的生命力。从 Web 技术的角度来看，总是希望有一种方法能够取代 Native（原生），事实上，这种趋势也越来越明显。

具体到本篇内容，分为 Block 的应用、网络请求、JavaScript 基础、Web 与 Native 的交互，这四部分并不是独立的，而是通过一条主线贯穿在一起的。

本质上讲，Web 与 Native 的交互是 Web 与 Native 技术的融合。没有网络请求，Web 无从谈起。而 iOS 的网络请求，必然用到 AFNetworking；既然是网络请求，就会用到异步调用，回调和 Block 是异步处理机制的基础，所以才出现了 Block 和网络请求两章。

从 iOS 自身技术来讲，似乎 Web 加载再简单不过了，不就是调用一个 UIWebView 吗？其实，技术水平的差异就体现在这个缝隙地带。我们是否具备 Web 的基础技能呢？是否懂得一些 JavaScript 呢？这些知识看似与 iOS 无关，实际上它们是密不可分的。如果 iOS 开发者不懂得 JavaScript，那么在涉及 Web 与 Native 交互时，沟通将会很吃力，以至于影响到产品开发的进度和团队的合作效率。

从技术层面上讲，Web 与 Native 的交互方式有多种，尽管苹果公司推出的 JavaScriptCore.framework 已有多年，但在项目开发中，还是有人在用传统的 HTTP 特殊字符拦

截方式。虽然也能满足项目的需求，殊不知，一个过时的框架技术，会对产品后期的维护埋下太多的坑。

本篇讲到的 JavaScript 基础，目的是指出，相比其他编程语言，JavaScript 有什么特别之处。

Web 与 Native 的交互，重点在讲述网页与 iOS 之间的相互调用。具体来说，Objective-C 如何调用 JavaScript，反之，JavaScript 又如何调用 Objective-C。而它们相互调用的桥梁是 JavaScriptCore.framework，前期的 Block 概念在这里得到充分的应用。

（3）全栈开发技术。

iOS 开发者在经历几年的积淀之后，常常会产生一种莫名的困惑，未来的技术路线何去何从？尽管 iOS SDK 博大精深，但作为一个 App 开发者来说，并没有那么多金矿可挖。历经十年的发展，iOS 的第三方框架已非常成熟，如果你的工作仅仅是为了开发一个 App，随着时间的推移，对技术路线的方向会愈发渴望。这个时候，有必要停下来，仔细规划一下了。

学无止境！问题是该怎么学。摆在 iOS 开发者面前的有两个选择：先熟悉 Objective-C，再向 Swift 深入，这是一种纵向学习方法，试图将 iOS 开发进行到底，这种方法适合开源框架的打造者，紧跟 Apple 技术更新的步伐；还有一种学习方法——横向发展，不仅掌握 iOS 技能，还需要扩展羽翼，学习更多的知识。IT 知识如同浩瀚的海洋，该如何撷取一朵适合自己发展的浪花呢？我们先来看看 iOS App 的生态环境，或许从中能得到答案。

无论是 iOS 还是 Android，App 原生开发模式的最大弊端是版本的迭代与升级的任务繁重。为了解决这个问题，才引入了 HTML5 技术。移动互联网产品，从产品形态来看，分为 iOS App、Android App、微信公众号（小程序）、后台管理页面、数据管理等；从开发的技术工种来看，分为 App（iOS、Android）工程师、前端工程师、后端工程师；从广义层面来看，App 也是前端的一种展现形式，App 开发离不开 Web 技术。从某种意义上说，前端=App+JavaScript。作为 iOS App 开发者，一旦掌握了 JavaScript，其技术路线的前景瞬间开阔了很多，而前景的方向就是“全栈技术”路线。

一个偶然机会，我接触到了全栈（Full Stack）的概念，并瞬间被它的理念所吸引。这里说的全栈，不是传统的 LAMP（Linux、Apache、MySQL、PHP），而是一种全新的以前端为主导的框架，所谓“大前端”、“全端”，指的是以前端为核心的框架。最终，我把框架选型聚焦在 MEAN（MongoDB、Express、AngularJS、Node.js）上。MEAN 全栈技术框架所用到的每个组件（MongoDB、Express、AngularJS 和 Node.js）都是基于 JavaScript 语言开发的，原本 JavaScript 是为网页设计的语言，但自从有了 Node.js 之后，前端工程师也可以写后台了。Node.js 让前端开发像子弹一样飞！

选用 MEAN 全栈技术，可以快速地实现开发，尤其是到了产品的运营阶段，其优势表现得非常明显。我们知道，今天的任何一款移动互联网产品，都离不开微信公众号的推广，大多

出彩的产品，在它的微信公众号内，所展示的是一套完整的业务逻辑，而不是几个简单的页面。这就是说，一个运营成功的产品，越来越倚重于全栈技术。

为此，本书在后半部分引入了全栈技术开发，从而进入全栈开发的世界。

全栈技术包括后台与前端。一说到服务器端开发，自然想起 Java、.Net、PHP，不过，近几年来，Node.js 风生水起，如果你想学习一门更具前沿技术的服务器端开发平台，Node.js 是一个不错的选择。

尽管 Node.js 自身已足够强大，但其生态系统的构建还要借助于 Express、AngularJS、MongoDB 以及模板引擎。既然 Express 是基于 Node.js 之上的后端框架，对初学者来说，借助 Express 框架更容易快速上手。

前端框架，本书选用了 AngularJS。在吹响全端号角的今天，我们越来越强调前端框架的重要性。在前端的世界，AngularJS 可谓“玉树临风”。在 MEAN 全栈中，Node.js 和 Express 负责后端处理，而与网页交互的正是 AngularJS，因此，可以想象 AngularJS 在前端框架地位之高。

把 MongoDB 数据库应用到 MEAN 全栈中，可谓相得益彰。通过 MongoDB，你对全栈开发会有一个完整的、全新的认知。

学习一门编程技术，最有效的途径还是实践。对于书中出现的每个知识点，都辅以相关的代码实例。书中的实例不是独立的，而是贯穿了整个全栈的技术点。

践行全栈之路

用了 MEAN 全栈，它到底能带来什么好处呢？这里，以我们发布的一款产品——“点时”App 为例。“点时”是一款轻量级的知识分享平台，内容以语音为主。这样的一款产品，从生态上讲，包括 iOS App、Android App、微信、后台的课程发布与运维管理。传统的做法是项目开发组分为前端与后台两套人马，要么前端等后端，要么后端等前端，而我们采用了 MEAN 全栈架构，不再区分前端与后台，开发效率明显提升。借助 MEAN 全栈框架，它带来的最大好处是减少了前后端之间的依赖，App 开发工程师可以通过全栈技术实现一次华丽的转身！

本书的源码

在学习本书示例代码时，可以按照书中讲解的步骤，一步一步地手工敲入所有代码，也可以下载随书所带的源码。本书所有的源代码都可以从 GitHub 下载。

源码下载地址：https://github.com/leopard168/iOS-Enterprise。

勘误和支持

我尽最大的努力确保正文和代码没有错误，但随着开发环境版本的变化，错误在所难免。

如果读者发现书中的任何错误，例如错别字或代码片段无法运行等，希望您能及时反馈给我。您提交的勘误不仅能帮助自己，还能让其他读者受益。

读者可以在下载源码的地方（GitHub）进行反馈，也可以通过下面的联系方式与笔者沟通。

致谢

参与本书编写的还有林志红、尹陆军、张俊、马钧君、和凌群、高宁、刘晓波、牛雪峰。在本书成稿的过程中，得到了很多人的指点和帮助。这里，特别感谢北方工业大学计算机学院，从学生到老师，都给我了很多的支持与帮助，每次审视这些教学案例，都会在技术上得到一次升华。

在本书出版之前，初稿曾在北方工业大学计算机学院试用。在整个教学过程中，学生们提出的很多疑惑，都在本书中得到诠释。

这里，特别感谢北方工业大学计算机学院院长马礼教授在百忙中抽出时间为本书作序。

感谢电子工业出版社，正是你们卓有成效的工作使我保持了敲击代码的激情。

作者交流方式

作者的 QQ：2385911707@qq.com。

作者的微信号：leopard2385911707。

作　者

2017 年 9 月

我的 iOS 授课经历

在过去的几年，我为北方工业大学计算机学院的大四学生讲授过几期 iOS 课程，在听课前，学生们已经具备了一定的编程基础，如 C、Java 语言等。每次开课都是从 iOS 入门讲起。每学期结束后，我都会对自己的教学心得稍做总结，感触颇深。

初学者习惯于敲击代码，对设计模式没有多大兴趣。

每当讲到代码部分，大家都容易理解，尤其是敲击代码阶段，轻车熟路。而让人感到困惑的是，一旦涉及 Storyboard 中的对象拖曳，瞬间一片茫然。仿佛对象的拖曳是天外来客，难道作为一名程序员，就一定要比拼代码量吗？

我们习惯于大谈特谈软件的设计模式，但又有多少人会有意识地将设计模式应用到自己编写的代码中呢？不错，MVC（Model-View-Controller）设计模式是最为主流的设计模式之一。如果真正领悟了 MVC 设计模式，就不会对 Xcode 拖曳对象感到别扭，相反，你会为这种 Ctrl-Drag 操作而叫绝。这是因为，MVC 设计思想在 Xcode 平台上表现得淋漓尽致！

对于软件开发者来说，MVC 设计模式是极其重要的。如果不善用 MVC，软件的后期维护将变成一场灾难。

从事软件开发的人都有这样的经历，很不习惯维护别人的代码。也就是说，代码的可维护性着实让人头疼。在我的工作中，需要经常走查他人编写的代码。最不想看到的是，代码中有通篇的、莫名的数字。对于 App 开发者来说，看到大量的控件坐标会着实令人不解。通过代码创建自定义的视图，会充斥大量的坐标数字。这是因为，创建一个视图（按钮、标签、编辑框），至少需要设置它的左上角（原点）的坐标，还有其宽度和高度。这样的代码，如果让别人来维护，简直让人抓狂！

如果善用 MVC 设计模式，这些痛苦将会迎刃而解！

MVC 中的 V（View）- C（Controller）关系，完美地体现在了 Storyboard 的对象拖曳上。如果真正理解了 MVC 的设计精髓，就不会为对象的拖曳操作而烦恼。对一个 App 来说，主要解决这样几个问题：页面（视图）的展示、页面（场景）的跳转和数据的交互。可以说，App 的开发，大部分的工作量都花在了页面的布局上。对于像 App 这类 UI 产品，我们自然希望有这样的一个可视化开发平台，做到所见即所得，需要哪个控件，就拖曳哪个控件，再根据产品

的设计要求，对页面上的控件进行布局。这样一来，通过简单的拖曳操作，一个完整的页面便会跃然纸上。回头一想，Xcode 不就是我们所期望的那个图形化开发平台吗？

在 Storyboard 或 Xib 上，拖曳一个对象（控件），从本质上讲，这个操作就是自动生成了一个视图（View）。至于视图要显示的具体内容，这是需要控制的。Xcode 通过创建视图的 IBOutlet 和 IBAction 来控制视图。ViewController，顾名思义，是视图控制器的意思。而视图控制器就是一个类，具体来说，就是一个.h 和.m 文件。在创建 IBOutlet 和 IBAction 时，初学者所遇到的困惑是，不清楚往哪个文件拖曳，也不清楚拖曳到文件的哪个部分。为了搞明白这个问题，需要弄清楚一个概念：只有该视图所对应的视图控制器，才能控制对应的视图；拖曳的方向是对应的.h 或.m 文件，拖曳的具体指向是@interface 与@end 之间。

为了熟记这种 Ctrl+Drag 操作，这里给出三步法。

第一步：拖曳，在 Storyboard 中，直接拖曳一个 UIViewController。

第二步：创建，创建该视图所对应的视图控制器类（UIViewController）。

第三步：关联，将拖曳的 UIViewController 与创建的视图控制器类关联起来。

尽管通过拖曳对象的方式可以省去很多不必要的代码，但这也仅仅是节省一部分代码而已。Storyboard 和 Xib 不是万能的，对于平台无法直接满足的定制化的控件，需要我们通过手动编码的方式来实现。

开发一款可用的、可运营的 App 产品，是一件很不容易的事，“生命，本不该浪费在一些无趣的事上”。我们应该把精力集中在产品的业务实现和用户体验上，而不是为 UI 的实现花费太多的时间。

忍不住也来说说 iOS 的设计模式

不管是什么软件开发平台，都会宣扬其设计理念，大谈特谈其设计模式。这里也忍不住想说说 iOS 的设计模式。

iOS 最基本的设计模式就是 MVC。

大多 MVC 参考书都是这样介绍的：MVC 是一种设计模式，M（Model）、V（View）、C（Controller）。Controller 控制视图的显示，Model 提供数据，Model 与 Controller 交互，但 Model 从不与 View 直接打交道。MVC 解决了视图与数据隔离的问题，降低了数据与视图的耦合性，便于维护。

对于 iOS 初学者来说，看完这段话，是没有感觉的。当我们学习某种模式时，首先要明白它讲的是什么？为什么用它？怎么用它？明白了这三点，那才是真正地明白了。

MVC 的概念，我们已经清楚了。不清楚的是：为什么要用 MVC？仅仅讲到数据与视图的分离，这是不接地气的。也就是说，大道理都明白，谈到具体的使用，仍是模棱两可。

如果不想再单独创建一个 Model 文件，直接把数据放在 Controller 中，难道这样不行吗？这还真的行。我们注意到，确实有些 App 就是这样做的。当 App 数据量很小时，体现不出设计模式的优势，想怎么用就怎么用，反正功能也实现了，而且，还减少了 Model 文件，感觉更容易些。

这里，我们所讨论的是较为复杂的 App。对于复杂的 App 来说，首要特征就是数据量庞大，数据的管理是一个难点（这里暂不讨论数据管理问题，后续章节有单独的介绍）。

做软件，总是希望能复用。对于 App 来说，复用有以下几种情况。

（1）数据的复用：如果新创建了一个项目，想重用之前 App 的数据。也就是说，换汤不换药。同一套数据，不同的 View。这个时候，就需要把之前的数据分离出来了，分离到一个独立的数据文件中。在新的项目中，直接加载这个数据文件就可以了（这里说的数据文件，不是某个 plist、sqlite 文件，而是 Model 下具有逻辑功能的文件）。

（2）视图的复用：同一套 UI 框架，只是展示的数据不同。这个时候，数据文件、视图（View）文件和视图控制器（View Controller）文件是完全独立的。

（3）Universal App：是指一个 App 可以根据终端自适应布局，既支持 iPhone 版，又支持 iPad 版，这样的 App 称之为 Universal App。既然是同一个 App，就应该共享同一数据。如果数据不是独立的模块，而是将数据混在了 View Controller 中，那么管理起来将非常不方便。难怪有 iOS 开发者抱怨，做一个 Universal App 还不如单独开发两个 App 简单：一个 iPhone 版 App，一个 iPad 版 App。如果真的是对同一个产品创建两个工程的话，后期维护的工作量可想而知。

目　录

iOS 基础篇