移动开发经典丛书

PEARSON

精通iOS框架(第2版)

[美] Kyle Richter　Joe Keeley　著
冯宗翰　江铭　朱倩　译

清华大学出版社

移动开发经典丛书

精通 iOS 框架

(第 2 版)

[美] Kyle Richter
Joe Keeley 著
冯宗翰 江铭 朱倩 译

清华大学出版社
北 京

图书在版编目(CIP)数据

精通 iOS 框架：第 2 版 / (美) 里克特 (Richter, K.)，(美) 姬莉 (Keeley, J.) 著；冯宗翰，江铭，朱倩 译. —北京：清华大学出版社，2016

(移动开发经典丛书)

书名原文：Mastering iOS Frameworks: Beyond the Basics, Second Edition

ISBN 978-7-302-43381-1

Ⅰ. ①精… Ⅱ. ①里… ②姬…③冯…④江… ⑤朱… 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(2016)第 075620 号

责任编辑：王　军　李维杰

装帧设计：牛艳敏

责任校对：成凤进

责任印制：宋　林

出版发行：清华大学出版社

网　　址：http://www.tup.com.cn，http://www.wqbook.com

地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084

社 总 机：010-62770175　　邮　　购：010-62786544

投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

印 刷 者：清华大学印刷厂

装 订 者：三河市少明印务有限公司

经　　销：全国新华书店

开　　本：185mm×260mm　　印　　张：29　　字　　数：742 千字

版　　次：2016 年 5 月第 1 版　　印　　次：2016 年 5 月第 1 次印刷

印　　数：1~3000

定　　价：79.80 元

产品编号：065611-01

译 者 序

苹果公司的产品一直在引领着智能设备的发展，苹果公司的伟大就在于它的每一个产品都臻于完美，让使用者赞叹不已。我也是一个不折不扣的“果粉”，不过我不满足于使用它的产品，我也希望自己能够编写出改变世界的产品。虽然这个理想可能比较大，不过正是抱着这个梦想的开发者们创造出了一个又一个神话，从“疯狂的小鸟”到 Snapchat，从微信到 Uber，无一不让使用者疯狂。智能手机已经改变了人们的生活习惯，“互联网+”的时代也已经正式到来，而苹果公司和它的 iOS 系统无疑是这个领域的佼佼者。一方面 iOS 平台因为它的封闭而饱受指责，另一方面也正是得益于这样的生态系统，保护了开发者的利益，让他们有更多的动力专注于创造更优秀的产品，这也正是 App Store 中的应用无论在界面上还是用户体验上都优于其他平台的原因之一。

我从事 iOS 开发也有将近 10 年的时间了，记得最初手头甚至没有一本像样的参考资料。可想而知，在学习开发技术的最初一段时间有多么痛苦和无助，唯一能够帮助到我的就是苹果给出的官方文档，不过全英文的解释极大延缓了我对技术的探索。不过随着 iPhone 产品越来越出色，越来越多的人开始注意到 iPhone 的潜力，也有越来越多的开发者转到 iOS 平台，市面上有关 iOS 开发的入门教程也越来越多了。随着一步一步走进 iOS 的世界，你会觉得入门教程已经不能满足你对技术的渴求了，而市面上真正介绍高级开发技术的书又很少，大多数教程也非常晦涩难懂。本书的作者对 iOS 开发有着非常丰富的经验，同样有着很深的理解。他们很准确地抓住了从各种框架的介绍入手这一切入点，正是这些框架的熟练使用才能让一个菜鸟慢慢成长为一个资深开发者。这一点我不得不佩服作者的智慧，在我自己学习 iOS 开发遇到瓶颈的时候，多么希望有这样一本参考书，在我遇到问题的时候可以直接拿来解决问题，不过那时候还没有这本书，而现在的读者就幸运多了。

本书不仅有着非常明确的切入点，在成书的过程中也有过人之处。我们知道，学习编程技术最好的办法就是亲自动手实践，本书也正是采用了这种方法让读者能够将理论和实践相结合，最终通过自己动手完成对应功能的开发来掌握相应的技术。另外本书的结构也很有特色，除了几章关联度很强外，几乎每一章都是独立的，你完全可以在需要某一技术或知识的时候直接翻到相应的部分，同时本书又几乎覆盖了 iOS 中高级开发技术中所有的框架，所以只要把这本书放在手边，就可以应对许多开发中常见的问题了。

能够有幸拜读本书并完成本书的翻译工作我觉得很幸运，就像是同这样一位 iOS 开发界

的大佬对话一般，怀着欣喜和紧张完成了本书的翻译工作，期间不敢有一点怠慢和疏忽，在这里要感谢清华大学出版社的编辑对我的帮助，他们为本书的翻译投入了巨大的热情并付出了很多心血。没有他们的帮助和鼓励，本书不可能顺利付梓。还要感谢无锡职业技术学院领导和同事们的帮助，正是他们对我的支持让我可以专心于本书的翻译工作。也谢谢我的爱人和家人，是她们的支持让我能够专心完成本书的翻译工作。虽然在翻译过程中力求“信、达、雅”，但是鉴于译者水平有限，错误和失误在所难免，如有任何意见和建议，请不吝指正。感激不尽！本书所有章节由冯宗翰翻译，参与本次翻译的还有江铭、朱倩、孙婷婷、许亦男、孙伟、冯树彪、张怀洲、李志锋、李爽等，在此一并谢过。

最后，我相信这本书一定会成为开发者身边一本不可或缺的参考书，而这也正是本书作者最大的希望，相信大家一定能创造出改变世界的产品，向着梦想前进！

冯宗翰

序　言

从 2008 年 iPhone SDK(现在叫 iOS SDK)的第一个测试版发布之日起，我就一直从事有关 iOS 的开发工作。那时我主要关注有关 Mac 桌面程序的开发，没有过多考虑移动应用的开发。

如果你希望成为一个早期开发者，那你只能靠自己了。苹果公司一贯的做法就是文档非常少，并且由于访问 SDK 需要 NDA——秘密解码环，因此最初你只能靠自己。你还不能在 Google 上或打开 StackOverflow 寻求到帮助，并且那时也一定没有任何介绍 SDK 的书籍。

从苹果发布最初的 iPhone 到现在已经走过漫漫 8 年时光(是的，真的只有 8 年)。iPhone SDK 现在也被称为 iOS SDK。有关 iOS 开发的书籍和博客，以及播客和研讨会层出不穷。从 2009 年起，WWDC 大会变得更加难以参加，使得新老开发者在学习最新的平台技术时变得越发困难。尤其对于 iOS 开发者，要学的新东西真的太多了。

作为一名 iOS 开发者，我遇到的最大的难题就是设法驾驭苹果工具箱(kit)中所有的组件和框架。iOS HIG 本能帮助到我们，不过它对于组件和框架的介绍还不够详细深入。现在我们确实能够通过 Google 或者结合 StackOverflow 找到资料，不过这些资料一般都仅解释了如何去做，很少探究为什么要那样做，通常也无法做到很详细地分析。

所以 Kyle 和 Joe 决定这样做——给出所有这些框架的详细介绍，让读者可以全面了解组成 iOS SDK 的核心框架。

很荣幸与 Kyle 和 Joe 相识多年。他们是我所见过的最聪明的开发者。这些年来他们都各自编写了大量优秀的应用，并通过分享他们的经验为 iOS 开发社区不断贡献自己的力量，不断在研讨会上发表出色演讲，出版有关 iOS 开发的书籍。如果你有任何关于 iOS 的问题，有机会得到 Kyle 和 Joe 的解答将会是一件很美妙的事情。

不过让他们如此优秀的原因还不仅是他们如百科全书一般的 iOS 开发知识，更在于他们愿意同遇到的每个人分享这些宝贵的知识。Kyle 和 Joe 没有竞争对手，有的只是朋友。

Kyle 和 Joe 对于 iOS SDK 的深入理解贯穿本书，这也是我喜欢本书的原因之一。本书对每个组件都进行了详细介绍，有些内容甚至在网上都很难找到。

我还非常喜欢本书的结构。读者不需要从头到尾阅读。相反，你可能因为需要了解如何实现集合视图而翻开本书，或者在想学习如何在后台线程上运行任务时打开本书。在你需要本书时只需打开它，找到解决办法，将其在自己的代码中实现，然后再把它放回书架上直到下次再遇到困难。这就是该书能够成为 iOS 开发者最重要的一本参考书籍的原因，不管你是

初学者还是经验丰富的老手。你可能觉得自己精通 Core Location 和 MapKit，不过我觉得你在本书中一定能找到以前没有接触过的知识。

Kyle 和 Joe 为人非常谦虚，从不骄傲自大。他们从来不认为自己比其他开发者优秀。他们将这种精神一点点灌输到 Mac 和 iOS 开发者社区，使其成为一个开发者之间互助学习的大平台，对这个行业的发展帮助很大。这本著作也是他们无私分享多年经验和知识的另一个见证。

本书同 Mark 和 LaMarche 及 Sadun 的著作一样，将始终伴随在我手边。当我在 2008 年第一次开始做 iOS 开发时我多希望有这么一本书。现在终于如愿以偿，有了这本书就方便多了。

— Kirby Turner

White Peak Software 公司首席程序员，*Learning iPad Programming: A Hands-On Guide to Building iPad Apps, Second Edition*(Addison-Wesley Professional)一书的作者，Cocoa 开发者社区的管理者和研讨会的热衷者。

作 者 简 介

Kyle Richter 是 MartianCraft 公司的 CEO，MartianCraft 公司曾赢得 Mobile Development Studio 称号。Kyle 在 20 世纪 90 年代初就开始从事软件开发工作，并始终专注于基于苹果公司平台的开发，他在 iOS 开发方面已经出版和共同出版了多种著作，包括 *Beginning iOS Game Center Development*、*Beginning Social Game Development* 和 *iOS Components and Frameworks*。利用管理 MartianCraft 公司每日运营的同时，Kyle 还要在全球出差，介绍有关开发和公司管理的经验。现在佛罗里达群岛是他的家，因为在那里他要花时间陪伴他可爱的边境牧羊犬。你也可以通过@kylerichter 在 Twitter 上找到他。

Joe Keeley 是 MartianCraft 公司的合伙人及首席工程师。Joe 为 iOS 客户项目提供技术指导并主导了许多成功的项目。他从 Apple Ⅱ开始就热衷于程序开发，在他的职业生涯中从事过许多不同技术和系统项目的开发。在美国，Joe 出版了多种有关 iOS 和 Mac 技术开发的参考书。Joe 和妻子及两个女儿居住在科罗拉多州丹佛市，在闲暇时间他喜欢从事击剑运动。他在 Twitter 上的名字是@jwkeeley。

前　言

欢迎阅读《精通 iOS 框架(第 2 版)》!

对于现在的读者来说，几乎有上百种“iOS 入门教程”可供选择，介绍特定专题的进阶教程也数不胜数，例如专门介绍 Core Data 知识或者专门介绍有关数据安全方面的教程。不过令人感到困惑的是还没有一种书能够作为初学者向更高级内容前进的桥梁。

撰写本书旨在向读者介绍中高级的开发知识，因为这些看起来零散的框架很多都无法单独出书。并不是这些框架不够吸引人，而是因为它们还不能算是一个很大的专题，不需要过多篇幅的介绍。从如何使用 JSON 到如何访问照片库，这些框架都是专业 iOS 开发者每天都会用到的，不过市面上少有书籍介绍它们。

此外，对很多进阶内容的讨论也要考虑到面对的读者可能是初学者。翻开一本 500 页的 Core Data 教程也是需要勇气的，不过还好本书第 15 章给出了一个简捷的 Core Data 入门教程。其他有关类似的入门教程还包括调试和工具、TextKit、HomeKit、HealthKit 和 CloudKit 等章节。

对于 Game Center 排行榜和成就榜、AirPrint、音乐库、地址簿和 Passbook 的内容则给出详细全面的讲解。无论你是刚刚完成第一个 iOS 项目开发的初学者还是一名经验丰富的资深开发人员，都可以从本书中找到对你有帮助的内容。

书中所有章节的内容都根据 iOS 8 版本进行了更新。如果你遇到有关兼容性的问题请联系我们，我们会发布更新并进行修改。

如果你对本书的内容有任何建议，或者发现并修改了本书的一些错误，它们都对本书后续版本有很大的帮助，可以通过 mastering.ios.frameworks@gmail.com 联系我们。我们非常愿意听取任何能使本书变得更加完善的建议，并会不断致力于让本书更加完美。

需要具备的知识

本书尽力让所有的示例程序和知识讲解更加简单易懂，不过它毕竟还是一本针对中高级开发者的参考书。所以为了更好地使用这本书，你需要具备基本的 iOS 开发知识，以及 Objective-C 和 C 语言的知识，熟悉 Xcode、Developer Portal、iTunes Connect 和 Instruments 工具的使用。学习 Objective-C 和 iOS 的内容时可以参阅 Stephen G. Kochan 撰写的 *Programming in Objective-C* 一书和 Maurice Sharp、Rod Strougo 及 Erica Sadun 共同撰写的 *Learning iOS Development* 一书。

准备工作

虽然可以借助 iOS 模拟器开发并测试 iOS 应用，不过我们还是建议你至少拥有一款 iOS 设备用于测试：

- Apple iOS Developer Account：iOS 开发工具 Xcode 和 iOS SDK 的最新版本可以从苹果公司网站上的 Developer Portal 进行下载(http://developer.apple.com/ios)。要在 App Store 上发布应用或者在一台个人设备上安装并测试应用，需要支付每年 99 美元的开发者账号费用。
- Macintosh Computer：要开发 iOS 程序并运行 Xcode，你需要一台能够运行最新发布的 OS X 系统的 Mac 机器。
- Internet Connection：iOS 开发中的很多功能都需要你的 Mac 机器和 iOS 设备保持网络连接。

本书结构

除了个别章节(Game Center 和 Core Data)之外，本书的每一章都是独立的。你可以按照顺序从头到尾阅读本书，也可以跳过一些你暂时不需要的专题而直接找到你需要的内容，我们撰写本书的目的就是使其成为一种能够解决大部分 iOS 开发任务的快捷参考书。

下面是各章简介：

第 1 章“UIKit Dynamics”：iOS 7 增加了 UI Kit Dynamics 元素，可以让 UIView 添加模拟物理运动的动画效果。你会学到如何给对象添加标准的动态动画效果、设置物理属性等功能。从重力效果到元素属性设置，按照从易到难的顺序介绍 7 种效果的实现方法。

第 2 章“Core Location、MapKit 和 Geofencing 框架”：iOS 6 带来了全新的苹果地图和相关数据。该章会介绍如何通过使用 Core Location 来确定设备的位置，如何在应用中显示地图，以及如何在地图上使用自定义标注、覆盖和弹出气泡。还会介绍如何设置地理监控区域(地理围栏)，当设备进入或离开一个区域时设备会有提醒。

第 3 章“排行榜”：Game Center 排行榜提供了一种非常简单的方式让你的 iOS 游戏或应用增加社交元素。该章会带来一个名为 Whack-a-Cac 的 iPad 游戏，我们为其添加一个排行榜。用户将会学到实现 Game Center 排行榜的所有步骤，同时也会对如何实现带有自定义界面的排行榜有所了解。

第 4 章“成就系统”：该章继续使用前几章介绍的 Whack-a-Cac 游戏。你将学习到如何在这个 iPad 游戏中实现成就系统。从如何使用 iTunes Connect 到显示成就进度，该章为你提供了所有快速创建成就系统所需的信息。

第 5 章“Address Book 框架初步”：很多项目现在都将地址簿整合到应用中，Address Book 框架是 iOS 系统最老的几个框架之一。该章你会学到如何使用这个框架，学习如何使用用户选取器、如何访问源地址数据以及如何修改并保存该数据。

第 6 章“Music Libraries 框架”：该章介绍如何在一个自定义应用中访问用户音乐库，包括如何查看具体音乐的信息数据以及如何从专辑中选择并播放一首乐曲。

第 7 章“实现 HealthKit 框架”：HealthKit 用于实现在应用间共享健康相关信息的功能。该章介绍如何开始使用 HealthKit，之后介绍如何访问 HealthKit 中的数据，以及如何读取和

写入不同类型的健康相关数据。

第 8 章“实现 HomeKit 框架”：该章介绍如何开始使用 HomeKit，它可以让 iOS 设备同智能家居设备进行通信。其中还对如何设置 HomeKit 进行了介绍，以及如何发现、开启设备和如何同这些设备进行互动，例如灯、锁或仓库大门的遥控器等。

第 9 章“JSON 的使用和解析”：JSON 的全称为 JavaScript Object Notation，是一种能够在不同平台和架构间传输数据的轻量级数据传输协议。所以它成为 iOS 客户端和服务器间传输复杂数据时最常用的协议。该章会介绍如何从一个现有的对象创建 JSON 以及如何将 JSON 解析为 iOS 对象。

第 10 章“通知机制”：iOS 支持两种类型的通知机制，分别是本地通知和远程通知。本地通知主要用于设备没有连接网络的情况；远程通知需要通过网络再经过苹果公司的 Push Notification Service 服务器推送通知到设备。该章会介绍两种通知机制的差异，并演示如何在一个应用中创建和使用这两种通知机制。

第 11 章“基于 CloudKit 的云存储”：CloudKit 提供公共的和私人的数据存储模式。该章会介绍基础的 CloudKit 概念，并通过创建一个应用来演示如何使用 CloudKit 实现远程存储和同步个人及公共数据的功能。

第 12 章“extension”：extension 机制提供了一种在应用沙盒之外访问应用功能的功能。该章介绍几种不同类型的 extension，并演示如何创建一个 Today extension 和一个 Apple Watch extension。

第 13 章“Handoff”：Handoff 是 iOS 8 和 Yosemite 最新引进的一种应用延续机制，它可以让用户在不同设备间进行切换，而应用无缝地在设备间进行延续。该章会介绍基础的 Handoff 机制，并演示根据开发者定义的 activity 和基于文档的 activity 实现 Handoff 功能。

第 14 章“AirPrint”：AirPrint 可以让用户通过无线方式在支持 AirPrint 的打印机上打印文档和图片，不过该框架属于 iOS 提供的众多框架中不太常用的。该章介绍如何在应用中快速有效地添加 AirPrint 打印功能。在该章的最后，你会学习到打印视图、图片、PDF 文档甚至渲染 HTML 文件。

第 15 章“开始使用 Core Data”：该章会介绍如何让应用使用 Core Data、如何设置 Core Data 模型，以及如何在应用中实现最常用的 Core Data 工具。如果你没有时间钻研那本 500 页的 Core Data 参考书，那么该章再适合你不过了。

第 16 章“使用社交框架整合 Twitter 和 Facebook”：整合社交元素是智能计算的未来，现在公认的做法是希望将所有应用都加入社交元素。该章会教你如何使用 Social 框架向应用中添加 Facebook 和 Twitter 功能。你会学到如何使用内置的整合功能创建新的 Twitter 和 Facebook 消息，还会学到如何从服务器获取反馈信息以及如何解析得到的数据。最后介绍如何使用该框架从自定义的用户界面上发送消息。完成该章的学习之后，你会对 Social Framework 有很深的理解，可以自如地在应用中添加 Twitter 和 Facebook 功能。

第 17 章“后台任务处理”：当应用离开前台运行时能够在后台继续完成某些任务是一个很重要的功能，这个功能在 iOS 4 版本中被加入进来，并随着时间的推移更加完善。该章会介绍当应用离开前台后是如何在后台运行的，以及如何执行 iOS 指定的一些特殊后台任务。

第 18 章“多线程开发的性能”：在主线程执行一些资源占用高的程序时可能会使系统变慢。该章会介绍一些由 Grand Central Dispatch 机制提供的方法，用于处理由于影响主线程性

能表现的复杂并发问题。

第 19 章“使用 Keychain 和 TouchID 保护并访问数据”：保护用户数据的安全性是一件非常重要的事，但有时却被粗心的开发者忽略了。几年前甚至有一家大公司因为使用明文对用户信用卡信息和密码进行存储而被大家批评。该章首选会介绍使用 Keychain 来保护用户数据，然后将开发涉及的安全性问题作为整体进行详细讲解。通过该章的学习你可以在用户设备上使用 Keychain 对任何小规模数据类型进行保护，这样用户就可以放心地使用这些数据了。

第 20 章“处理图片和过滤器”：该章首先介绍一些基础的图片处理技术，之后介绍一些有关如何应用过滤器的高级 Core Image 技术。示例程序对所有 Core Image 提供的方法给出了演示，并创建过滤链来实现实时交互功能。

第 21 章“集合视图”：集合视图是 iOS 6 中引入的一个功能强大的 API，可以让开发者更加灵活地处理可滑动界面的布局以及基于单元格的内容排版。除了新的布局功能之外，集合视图还提供了精彩的动画效果，在集合视图内容的淡入淡出及视图间的切换时都可以添加动画效果。

第 22 章“TextKit 介绍”：iOS 7 中引入了 TextKit 框架，它非常容易上手，是对 Core Text 非常重要的扩展。TextKit 可以让开发者在应用中提供非常丰富且互动性很强的字体格式。虽然 TextKit 是一个比较大的话题，不过该章会介绍一些基础的知识并实现一些常见的功能，例如从环绕图片添加文字到内嵌自定义字体属性。完成该章的学习之后，其实是为后面更深入地学习 TextKit 打下非常坚实的基础。

第 23 章“手势识别”：该章会介绍如何在应用中使用手势识别。同直接处理和解析触碰数据不同的是，手势识别对象可以提供一个简单且干净的方法来识别一些常见的手势动作并给予反馈。此外，还可以自定义手势并使用手势识别对象对其进行识别。

第 24 章“访问照片库”：现在 iPhone 实际上已经成为最流行的照相机，看看 Flickr 上用户上传的大量图片就可以证明这一点。该章会介绍如何访问用户照片库，并在自定义应用中处理这些照片和视频。示例程序演示了如何创建一个和 iOS 8 版本的 Photos.app 类似概念的应用。

第 25 章“Passbook 和 PassKit”：iOS 6 中苹果公司引入了 Passbook，它是一个独立的应用，用于保存用户的各种“卡”，或者诸如飞机票、优惠券、会员卡和演唱会门票等凭证。该章会介绍如何设置通行证，以及如何创建并发布它们，还有如何在应用中同它们进行交互。

第 26 章“调试和工具”：程序开发最重要的一部分工作就是能够调试和优化软件。很少有书籍会介绍这方面的内容，即使是简单的介绍也很少见。本章为你讲解了在 Xcode 中如何调试程序以及使用工具分析程序的性能。首先我们讲述了计算机错误的历史，之后逐步介绍了有关一些常见错误的提示和识别方法。然后简单介绍了断点的使用和调试器命令，并使用 Time Profiler 和 Leaks 工具分别对程序代码和程序内存的使用情况进行了分析。学习完该章后，你将对如何在模拟器和真机设备上定位程序错误及调试 iOS 应用打下坚实基础。

示例程序

本书的每个章节都是独立的(除了个别章节)，同时除了第 26 章“调试和工具”之外，所

有的章节都具有自己的示例程序。第 3 章“排行榜”和第 4 章“成就系统”使用同一个示例项目，基于这个项目演示各自的功能。每章都会对示例程序进行简单的介绍，并逐步让读者理解示例程序中包含的一些复杂内容，而不是直接介绍本章的理论内容。

我们致力于让示例程序简单易懂，所以有些代码可能没有进行很好的优化，可能也不是解决特定问题的最优方法。所以在每章内容中我们都会指出一些在实际创建应用时哪些步骤是不合适的。示例程序不是独立的应用，它主要用于演示相应章节所介绍的功能。本书的示例程序被特意设计为通用程序，所以读者应该关注章节中介绍的主要内容而不是那些同相应章节无关的代码。为将一些非必要的组件从示例程序中删除，我们进行了大量的工作，同时将代码行尽可能压缩。

很多读者看到示例程序中的代码可能会觉得意外，因为没有使用 Swift 语言，而是用的 Objective-C，本书就是这样设计的。因为所有的 API 都是用 Objective-C 写的，所以使用 Objective-C 可以更加容易地进行互动，而使用 Swift 还需要额外添加解释起来比较复杂的层。如果读者习惯使用 Swift 后，可以很容易地将概念移植到 Swift 中。示例程序前缀为“ICF”，大部分示例程序的名字都是章节标题名。

当学习 Game Center 章节时，bundle ID 关联了我们个人苹果账号的真实应用，以确保例子能够继续运行。这个例子还可以让读者学习到当开发者对同一个示例程序进行互动时会生成多个用户数据。对于 iCloud、推送通知和 Passbook 这 3 章，应用需要的设置都在相应的章节内容中进行了全面介绍，要在实际工作中使用这些程序，读者必须为自己的开发者账户创建一个新的 App ID 才可以。

获取示例程序

要随时获取最新版本的源代码，可以访问 https://github.com/dfsw/icf 网站下面的 Mastering 文件夹，也可以访问 http://www.tupwk.com.cn/downpage。这些代码都是面向公众的且是开源的。源代码按照章节分成各自的压缩文件，包含 Xcode 项目，每个章节都只包含一个项目。我们鼓励读者提供有关源代码的反馈信息并提出建议，这样我们就可以让工作更加严谨，并在本书出版之后不断对其进行改进。

安装并使用GitHub

Git 即版本控制系统，多年来深受开发者的喜爱。要使用 GitHub 上的代码，你首先需要在 Mac 上安装 Git。在 Xcode 命令行工具安装包中包含一个命令行版本的 Git，或者你还可以通过 http://git-scm.com/downloads 地址找到 Git 的安装文件。此外，Git 还有许多前端 GUI 形式，甚至有一个就是用 GitHub 开发的，这样可以更好地帮助开发者了解 Git，避免命令行晦涩难懂。如果你不想安装 Git，GitHub 也允许你以压缩格式下载源文件。

用户可以在 https://github.com/signup/free 上免费注册 GitHub 账户。Git 安装好之后，在终端程序中输入命令行$git clone git@github.com: dfsw/icf.git 就可以将一份源代码的拷贝下载到当前工作目录中。这一版书中用到的示例程序都在 Mastering 文件夹下。欢迎大家对示例代码提出修改建议。

联系作者

如果关于本书有任何意见或问题，可以通过 mastering.ios.frameworks@gmail.com 邮箱与我们联系，或者在 Twitter 上@kylerichter 和@jwkeeley。

致谢

要感谢所有为了本书默默在幕后付出的人，如果没有他们就没有这本书，虽然封面上的作者只有两个人的名字，不过这本书是通过大家的帮助才顺利完成的。首先要感谢的是 Trina MacDonald，如果不是她的领导才能和对我们的不断督促，我们可能永远无法完成本书的撰写。Pearson 的编辑们也给了我们极大的帮助，正是他们的不断努力，从一点小的错误到技术问题都逐页进行勘察，使本书变得更加完美。NiklasSaers、Olivia Basegio、Justin Williams、Sheri Replin、Elaine Wiley、Cheri Clark、ChutiPrasertsith 和 Gloria Shurick 也为本书的完成做出了贡献。

还要感谢 Langille Design (http://jordanlangille.com)工作室的 Jordan Langille，他为我们提供了第 3 章和第 4 章 Whack-a-Cac 游戏的设计方案。正是他的帮助让 Game Center 示例项目变得更加引人入胜。

不止我们自己为了撰写本书付出了大量的时间，家人和同事同样为此付出了许多时间。我们要感谢身边所有人的付出，让我们能有时间专注于本书的编写。

最后，还要非常感谢开发社区的朋友们。我们经常在开发者论坛上和博客里咨询大家问题，谢谢大家的提问和无私的反馈。如果不是 iOS 开发社区中大量参与者的努力，本书很可能就无法完成了。

目　录

第1章

UIKit Dynamics

苹果公司在 iOS 7 版本中引入了 UIKit Dynamics 框架，使用该框架，开发者可以很容易地将真实的物理模拟动作应用在 UIView 上。在之前的版本中，开发者只能将这种真实的动作效果整合到部分程序中，比如可滑动的单元格和下拉刷新动画等。苹果公司在 iOS 7 和 iOS 8 版本中向前迈进了一大步，将这些动画加入到核心 OS 库中，同时也在极力鼓励开发者使用它们实现动画效果。

UIDynamicItem 协议和支持该协议的动态元素使用户体验得到了极大提升。在程序中添加重力、碰撞、弹跳、瞬间位移等效果变得异常简单。介绍这些动态元素的 API 很简单，且很容易实现，应用这些功能提升用户体验易如反掌。

1.1 示例程序

示例程序(如图 1-1 所示)是一个基本的表格元素，用来展示 UIKit Dynamics 各种不同的功能。在这个程序中一共展示了从重力感应到属性设置等 7 个效果，每个效果都会在后面小节中详细展开介绍。除了表视图和基本的导航视图，示例程序并不包含任何专门针对 UIKit Dynamics 的功能。

注意

在同一个视图中使用 UIKit Dynamics 和自动布局可能会导致一些布局问题。通常，这是由于自动布局与 UIKit Dynamics 争抢视图上的正确位置，导致视图出现无法预料的错位而致。如果视图没有像预想那样呈现，开发者可以检查自动布局的相关设置来查看是否出现了冲突。

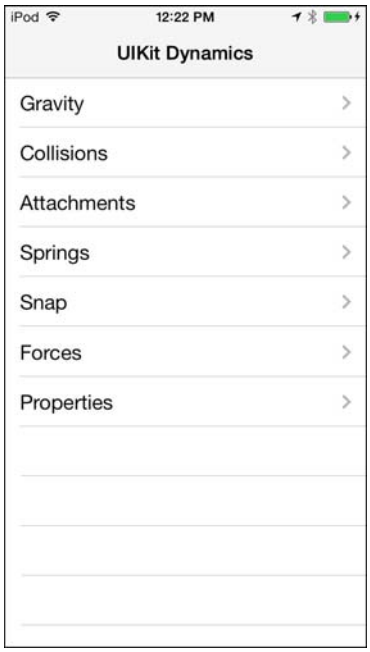

图 1-1　简单查看一下用于展示 UIKit Dynamics 各项功能的示例程序

1.2　UIKit Dynamics 介绍

UIKit Dynamics 是一组新的类和方法，最初是在 iOS 7 版本的 iDevices 中引入的。简单来说，UIKit Dynamics 通过在 UIView 视图中整合现实中的一些行为，提供了一种易于实现的方法来提升应用的用户体验。用最简短的术语来解释 UIKit Dynamics，其实它就是 UIKit 的基础物理引擎，不过它并不像传统的物理引擎一样是专为游戏开发而设计的。苹果公司提供了一些游戏框架，其中都包含了物理引擎，比如 SpriteKit。

当程序创建一个新的 UIDynamicAnimator 并将其添加到 UIView 中时，动态行为就会被激活。每个动画元素都可以对其属性和行为进行自定义，比如重力、碰撞检测、密度、摩擦力以及下面小节中将介绍的额外一些属性。

一共有 6 个附加类可以支持 UIDynamicAnimator 元素的自定义设置，分别是 UIAttachmentBehavior、UICollisionBehavior、UIDynamicItemBehavior、UIGravityBehavior、UIPushBehavior 和 UISnapBehavior。每个元素都允许指定自定义属性并且会在相应的视图中以真实的行为和动画反映出来。

1.3　UIKit Dynamics 具体实现

创建一个新的动画并将它添加到一个视图中，只需要两行代码就可以实现上述操作。示例中 self.view 对象即为将要使用 UIKit Dynamics 行为的对象。每一个特定的动态元素必须使用 addBehavior:方法添加到动画对象中。

```
UIDynamicAnimator *animator = [[UIDynamicAnimatoralloc]
initWithReferenceView:self.view];
```

```
[animatoraddBehavior:aDynamicBehavior];
```

每一个 UIDynamicAnimator 都是独立的，多个动画对象可以同时运行。对于一个持续运行的动画对象，对其的引用必须有效。当动画对象上的所有元素都处于静止状态时，动画对象此时不执行任何计算且处于暂停状态，不过实际操作中建议将不再使用的动画对象移除。

【游戏开发者的经验】

物理模拟对于游戏开发者而言已经使用了很多年了，很多难学的课程都已经学过了。现在物理层的处理技术已经蔓延到普通应用的开发中，下面介绍一些每位开发者都可以从中获益的基本原则。

当向游戏或应用添加物理特性时，请采取小步推进的方式。在多个互动代码段中试图找到出现的错误几乎是不可能的，采用越小步骤得到最终结果，程序也就越容易优化和调试。

在物理层进行处理时，有一些限制和边界在计算机模拟中无法体现。在 1997 年发布的经典游戏“死亡赛车”(Carmageddon)中，物理层处理是基于无上限帧率的。当计算机的处理速度变得越来越快后，帧率得到了大幅提升，在特定的公式中通过创建变量可以得到意想不到的结果。当把任何一种计算类型运用到物理引擎中时，需要确保其最大值和最小值都是符合要求且经过测试的。

预见下面这种意外情况：处理碰撞事件时，当 30 个对象发生重叠后，结果就会变得很扭曲。UIKit Dynamics 可以很好地确保开发者不会出现类似对象超过边界等情况，在处理上述碰撞场景时也能很完美地加以解决。不过当处理许多对象的复杂操作时也不能完全保证不出现边界情况和 bug。随着使用物理引擎的增加，越来越需要进行测试和调试，要能够预料到那些不期而遇和非常规情况下应该遵循的物理定律。

1.3.1　重力效果

重力效果被认为是最容易实现的 UIDynamicItem，同时也是实践中最常用的。苹果公司在 iOS 8 中重点强调了对重力相关元素的使用，用户操作重力相关的互动操作不需要再经过锁屏界面了。在 iOS 8 锁屏界面中，使用 UIGravityBehavior 函数可以实现向上拖动照相机图标并在到达中点之前松开手指，使屏幕再次返回锁屏状态的功能。在之前的 iOS 7 版本中引入 UIKitDynamics 时，这一功能还只能通过手工撰写计时器和传统动画相结合的方式才能实现。

下面的代码片段将在 frogImageView 视图中设置重力效果，该视图是 self.view 的子视图。首先为需要呈现动画的封闭视图创建一个新的 UIDynamicAnimator 对象，本例中呈现动画的视图即 self.view。创建一个新的 UIGravityBehavior 对象并初始化，初始化数组为使用重力效果的视图集合。设置本例中的重力行为参数：y 轴向下力度为 0.1。行为参数设置完成后，使用 addBehavior:方法将其添加到 UIDynamicAnimator 对象。

```
animator = [[UIDynamicAnimatoralloc] initWithReferenceView:self.view];

UIGravityBehavior* gravityBehavior = [[UIGravityBehavioralloc]
➥initWithItems:@[frogImageView]];
```

```
[gravityBeahviorsetXComponent:0.0f yComponent:0.1f];
[animatoraddBehavior:gravityBehavior];
```

注意

动态元素必须作为引用视图的子视图；如果动态元素不是子视图，动画对象就不会移动。

UIKit Dynamics 使用自身的物理系统，苹果工程师将其戏称为 UIKit Newtons。虽然它同标准的公式没有直接的关联，苹果公司还是做到了非常近似的效果。力度 1.0 大致等于 9.80655 m/s^2，即地球的重力。要使用地球重力十分之一的力，也就是 0.1 的力度。在 UIKit Dynamics 框架中使用重力效果不需要特别指定方向，默认就是向下的重力。如果 yComponent 参数为负值，重力方向才是向上的。同样，重力可以指定为沿 X 轴。元素还有密度属性，我们会在后面的 1.3.7 节中详细进行介绍。

运行重力效果示例代码，结果是 imageView 视图以大约十分之一地球重力的速度滑落(如图 1-2 所示)，并且完全滑出了屏幕。这是因为我们没有设置边界或碰撞事件，对象并不知道要碰到哪个边界才能停止运动，所以就一直滑落下去。

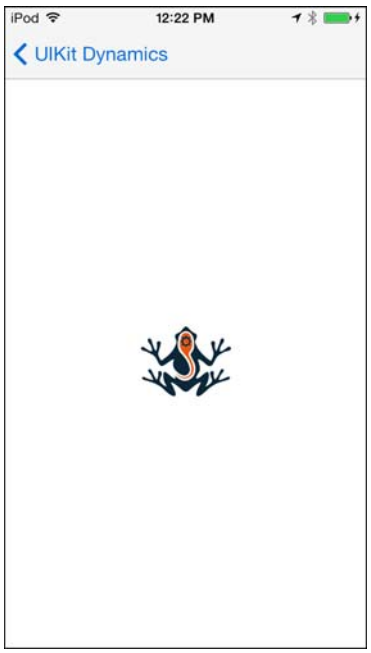

图 1-2　带有重力效果的图片视图在重力效果示例中向下滑到屏幕底端

1.3.2　碰撞效果

上一节中介绍了重力，不过应用了重力的对象会从屏幕底端滑出并一直无限地滑下去。这是因为我们没有在对象下降过程中设置碰撞点来阻止它。

我们对上一个示例程序进行修改，对封闭视图添加碰撞边界，同时添加第二个图片对象。碰撞示例的开始部分同重力示例一样，不过这里使用了两个图片视图。

创建 UICollisionBehavior 对象的过程同创建 UIGravityBehavior 对象的过程类似。对象通过将要应用效果的一些 UIView 视图进行初始化，本例中即两个 UIImageViews。除了视图之外，碰撞行为还需要在以下三个值中指定一个值作为参数：使用 UICollisionBehaviorModeItems 参数会使元素相互碰撞；使用 UICollisionBehaviorModeBoundaries 参数，元素虽然不会发生碰撞，但是会和边界发生碰撞；使用 UICollisionBehaviorModeEverything 参数会使元素在相互之间和同边界都发生碰撞。

要想程序中的对象和边界发生互动关系，需要先对边界进行定义。最简单的定义方法就是通过在 UICollisionBehavior 对象上设置一个称为 translatesReferenceBoundsIntoBoundary 的布尔值参数，在示例中我们将其用于 self.view 对象。边界也可以被设置为遵循某种路径，使用 addBoundaryWithIdentifier:forPath:方法使其遵循 NSBezierPath 表示的路径，或者使用 addBoundaryWithIdentifier:fromPoint:toPoint:方法使其遵循基于两个点表示的路径。

```
animator = [[UIDynamicAnimatoralloc] initWithReferenceView:self.view];

UIGravityBehavior* gravityBehavior = [[UIGravityBehavioralloc]
➥initWithItems:@[frogImageView, dragonImageView]];

[gravityBehaviorsetXComponent:0.0f yComponent:1.0f];


UICollisionBehavior* collisionBehavior = [[UICollisionBehavioralloc]
➥initWithItems:@[frogImageView, dragonImageView]];

[collisionBehaviorsetCollisionMode:UICollisionBehaviorModeBoundaries];

collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;

[animatoraddBehavior:gravityBehavior];
[animatoraddBehavior:collisionBehavior];
```

UICollisionBehavior 还提供了一个代理回调用于遵照 UICollisionBehaviorDelegate 协议。

```
collisionBehavior.collisionDelegate = self;
```

UICollisionBehaviorDelegate 函数有 4 个回调方法，两个用于碰撞开始，另两个用于碰撞结束。每一个回调集都有一个方法用来定义边界是否出现碰撞。所有方法都提供了对导致回调方法触发的对象的引用。检测碰撞开始的方法还提供了一个 CGPoint 对象，用于记录碰撞发生的确切位置。示例代码会在对象检测到碰撞时更新标签的显示。

```
-(void)collisionBehavior:(UICollisionBehavior *)behavior
➥beganContactForItem:(id<UIDynamicItem>)item
➥withBoundaryIdentifier:(id<NSCopying>)identifier atPoint:(CGPoint)p
{
    if([item isEqual:frogImageView])
         collisionOneLabel.text = @"Frog Collided";
    if([item isEqual:dragonImageView])
```

```
            collisionTwoLabel.text = @"Dragon Collided";
    }


    -(void)collisionBehavior:(UICollisionBehavior *)behavior
    ➥endedContactForItem:(id<UIDynamicItem>)item
    ➥withBoundaryIdentifier:(id<NSCopying>)identifier
    {

       NSLog(@"Collision did end");
    }
```

1.3.3 附着效果

附着效果定义了两个对象间的动态连接，可实现两个移动对象间的绑定关系。默认情况下，UIAttachmentBehaviors 都是固定在对象中心的，附着点可以通过程序自行设置为对象上的任何一点。

本节示例程序的创建以上一节“碰撞效果”中的程序为基础。仍然使用两个图片视图，边界碰撞已经创建好，并且已应用到 UIDynamicAnimator 对象上。创建一个新的 CGPoint 并设置青蛙图片视图的中心点作为其关联点。创建一个新的 UIAttachmentBehavior 对象并使用 initWithItem:attachedToAnchor:对其进行初始化。这里仍然需要对 UICollisionBehavior 进行额外的初始化，从而指定具体的点和其他对象的规范。将碰撞效果和附着效果都添加到动画对象。

```
animator = [[UIDynamicAnimatoralloc] initWithReferenceView:self.view];

UICollisionBehavior* collisionBehavior = [[UICollisionBehavioralloc]
➥initWithItems:@[dragonImageView, frogImageView]];

[collisionBehaviorsetCollisionMode:UICollisionBehaviorModeBoundaries];

collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;

CGPointfrogCenter = CGPointMake(frogImageView.center.x,
➥frogImageView.center.y);

self.attachmentBehavior = [[UIAttachmentBehavioralloc]
➥initWithItem:dragonImageViewattachedToAnchor:frogCenter];

[animatoraddBehavior:collisionBehavior];
[animatoraddBehavior:self.attachmentBehavior];
```

这些对象现在已经被一个长度为其初始距离的可见连接线围住，如果青蛙图片视图移动，那么龙图片视图将会保持中心点不变而随之移动。不过现在青蛙图片还不具有移动的能力，为解决这个问题，示例程序需要实现一个简单的平移手势。当青蛙图片视图在主视图中移动时，中心点位置更新的同时会设置更新的锚点。

```
-(IBAction)handleAttachmentGesture:(UIPanGestureRecognizer*)gesture
{
    CGPointgesturePoint = [gesture locationInView:self.view];

    frogImageView.center = gesturePoint;
    [self.attachmentBehaviorsetAnchorPoint:gesturePoint];
}
```

在移动过程中，碰撞边界始终有效并且覆盖了附着效果，这一点可以通过将龙图片移到视图边界上进行验证。

为了改变两个对象间的附着距离，还可以对附着视图的长度属性进行更新。附着点自身不需要是对象的中心点，可以通过调用 setAnchorPoint 方法设置任意偏移量作为附着点。

1.3.4　弹跳效果

弹跳效果(如图 1-3 所示)是上述附着效果的扩展。UIKit Dynamics 框架可以在 UIAttachmentBehavior 上额外设置一些属性，比如频率和阻尼等。

图 1-3　将弹跳效果应用在龙图片和青蛙图片上，展示了重力效果和设置 UIAttachmentBehavior 阻尼、频率的效果

下面小节的示例程序在创建 UIAttachmentBehavior 对象后又对其添加了三个新的 UIKit Dynamic 属性。第一个是 setFrequency，用于设置对象的振幅或摆动大小；第二个是 setDamping，用于校平动画峰值；第三个是 setLength，该属性也是根据其初始位置进行调整的。为更好地展示上述行为效果，我们仍然在例子中添加了重力效果。

```
animator = [[UIDynamicAnimatoralloc] initWithReferenceView:self.view];
```

```
UICollisionBehavior* collisionBehavior = [[UICollisionBehavioralloc]
➥initWithItems:@[dragonImageView, frogImageView]];

UIGravityBehavior* gravityBeahvior = [[UIGravityBehavioralloc]
➥initWithItems:@[dragonImageView]];

CGPointfrogCenter = CGPointMake(frogImageView.center.x,
➥frogImageView.center.y );

self.attachmentBehavior = [[UIAttachmentBehavioralloc]
➥initWithItem:dragonImageViewattachedToAnchor:frogCenter];

[self.attachmentBehaviorsetFrequency:1.0f];
[self.attachmentBehaviorsetDamping:0.1f];
[self.attachmentBehaviorsetLength:100.0f];

[collisionBehaviorsetCollisionMode: UICollisionBehaviorModeBoundaries];

collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;

[animatoraddBehavior:gravityBeahvior];
[animatoraddBehavior:collisionBehavior];
[animatoraddBehavior:self.attachmentBehavior];
```

现在，在屏幕中移动青蛙图片会使龙图片由底部上行 100 像素并按照设置好的附着效果和重力效果进行摇摆。

1.3.5 瞬间位移

元素可以在视图中动态地瞬间位移至另一个点，该功能非常容易实现。在示例程序中，动作已经绑定了一个点击手势，点击屏幕中的任何位置都会使指定图片瞬间位移到指定的锚点。每个 UISnapBehavior 一次仅能关联一个单独的元素，并且在初始化过程中元素终点的位置已被设置好。另一个属性阻尼系数也可以被指定，用于影响动作的移动速度。

```
CGPoint point = [gesture locationInView:self.view];
animator = [[UIDynamicAnimatoralloc] initWithReferenceView:self.view];

UISnapBehavior* snapBehavior = [[UISnapBehavioralloc]
➥initWithItem:frogImageViewsnapToPoint:point];

snapBehavior.damping = 0.75f;
[animatoraddBehavior:snapBehavior];
```

1.3.6 推力效果

UIKit Dynamics 还支持对力度的运用，比如推力。UIPushBehavior 的使用比之前的几种动作效果稍微复杂一些，不过同其他物理引擎相比还是很容易使用的。示例仍然使用了一个前面测试程序中用到的 UICollisionBehavior 对象，这确保了在推力效果作用时图片视图始终

处于屏幕范围内。

创建一个新的 UIPushBehavior 并使用一个图片视图初始化它。目前方向和加速度两个属性的值为 0.0。

示例程序还在屏幕中心用一个小黑方块的形式描绘了一个引用。

```
animator = [[UIDynamicAnimatoralloc] initWithReferenceView:self.view];

UICollisionBehavior* collisionBehavior = [[UICollisionBehavioralloc]
➥initWithItems:@[dragonImageView]];

collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;
[animatoraddBehavior:collisionBehavior];

UIPushBehavior *pushBehavior = [[UIPushBehavioralloc]
➥initWithItems:@[dragonImageView]
➥mode:UIPushBehaviorModeInstantaneous];

pushBehavior.angle = 0.0;
pushBehavior.magnitude = 0.0;

self.pushBehavior = pushBehavior;
[animatoraddBehavior:self.pushBehavior];
```

如果现在运行项目，图片视图将会始终固定在屏幕上，因为还没有对推力效果设置任何参数。创建一个新的平移手势，在其关联动作中会计算出 magnitude 与 angle 的新值并应用其中。在该例中，我们计算出一个角度，用于确定推力的来源。角度是基于中间参考点的，运动距离还要根据持续增加的力度进行计算。点击黑色方块的外面区域将会有一个力沿相应的方向作用于图片视图。图片离得越远，力度就越大。

```
CGPoint point = [gesture locationInView:self.view];

CGPoint origin = CGPointMake(CGRectGetMidX(self.view.bounds),
➥CGRectGetMidY(self.view.bounds));

CGFloat distance = sqrtf(powf(point.x-origin.x, 2.0)+powf(point.y-
➥origin.y, 2.0));

CGFloat angle = atan2(point.y-origin.y, point.x-origin.x);
distance = MIN(distance, 100.0f);

[self.pushBehaviorsetMagnitude:distance/100.0];
[self.pushBehaviorsetAngle:angle];

[self.pushBehaviorsetActive:YES];
```

除了可以手动设置角度和加速度的值之外，还可以对指定的目标点使用 setTargetPoint:forItem:方法自动进行计算。有必要对视图中的一部分施加力的效果，受力点并不一定是对象中心点，使用 setXComponent:yComponent:方法可以指定一个 CGPoint 类型的点作为受力点。

有两种类型的推力可以应用——UIPushBehaviorModeContinuous 和 UIPushBehaviorMode-Instantaneous，持续的力会推动对象不断加速，瞬间的力则会直接作用到对象上。

1.3.7　元素属性

动态元素有许多默认的预设属性，这些属性都可以自定义设置，用来表示对象针对物理引擎的不同响应。示例程序(如图 1-4 所示)展示了对一个图片视图修改默认属性和对另一个图片视图保留默认属性的对比效果。

图 1-4　修改动态元素的默认属性，展示在同一力的作用下得到的不同物理响应

要修改对象的属性，首先需要创建一个新的 UIDynamicItemBehavior 并使用相应的视图对象初始化它。结果就是其中一个对象动起来像一个橡皮球，对其应用了重力和碰撞效果后更容易弹跳。具体的属性和描述参见表 1-1。

```
animator = [[UIDynamicAnimatoralloc] initWithReferenceView:self.view];

UIGravityBehavior* gravityBeahvior = [[UIGravityBehavioralloc]
➥initWithItems:@[dragonImageView, frogImageView]];

UICollisionBehavior* collisionBehavior = [[UICollisionBehavioralloc]
➥initWithItems:@[dragonImageView, frogImageView]];

collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;

UIDynamicItemBehavior* propertiesBehavior = [[UIDynamicItemBehavior
➥alloc] initWithItems:@[frogImageView]];

propertiesBehavior.elasticity = 1.0f;
```

```
propertiesBehavior.allowsRotation = NO;
propertiesBehavior.angularResistance = 0.0f;
propertiesBehavior.density = 3.0f;
propertiesBehavior.friction = 0.5f;
propertiesBehavior.resistance = 0.5f;

[animatoraddBehavior:propertiesBehavior];
[animatoraddBehavior:gravityBehavior];
[animatoraddBehavior:collisionBehavior];
```

表 1-1　UIDynamicItem 属性及其描述

属　　性	描　　述
allowsRotation	一个布尔值，用于指定元素是否根据力度进行旋转，默认值是 YES
angularResistance	一个位于 0.0 到 CGFLOAT_MAX 范围内的 CGFloat 值，用于指示带有角度的阻尼值，这个数值越大，旋转的减速就越快
density	用于表示密度。100×100 对象的默认密度值为 1.0，100×200 对象的默认密度值为 2.0。调整密度的大小会影响重力和碰撞效果的反映
elasticity	有效值介于 0.0 到 1.0，用于指示当对象间发生碰撞时弹力的大小。0.0 表示不发生弹跳，1.0 表示整个力度会应用到碰撞对象上
friction	当两个元素相互滑动时的线性阻力，0.0 表示无摩擦力，1.0 表示最大摩擦力。此外，也可以使用大于 1.0 的值表示额外的阻力
resistance	开放空间中遇到的线性阻力，取值范围为 0.0 到 CGFLOAT_MAX。0.0 表示无阻力，1.0 表示当没有力作用于元素时元素应该停止移动

1.4　深入了解 UIDynamicAnimator 和 UIDynamicAnimatorDelegate

本章前面部分介绍了 UIDynamicAnimator，并且示例程序中都用到了 addBehavior 方法，不过这个类的强大功能远不止这些。除了可以添加动态效果之外，还可以每次移除一个效果或者对一组对象使用 removeBehavior:和 removeAllBehaviors 方法。要得到当前 UIDynamicAnimator 对象所有的动作行为，可以通过行为属性返回的数组进行查看。

还可以通过运行属性来查看动画的运行状态，可以使用 elapsedTime 值确定动画时长。UIDynamicAnimator 还带有一个关联的委托函数，即 UIDynamicAnimatorDelegate。该委托函数给出了两个方法用于处理暂停和重启动作。开发者无法显式暂停 UIDynamicAnimator 对象的运行，当所有元素都处于静止且不再发生运动时动画效果会自动停止。当将任何新的动作效果应用在元素上时，都会使其开始运动并返回激活状态。

```
-(void)dynamicAnimatorDidPause:(UIDynamicAnimator *)animator
{
    NSLog(@"Animator did pause");
}
```

```
-(void)dynamicAnimatorWillResume:(UIDynamicAnimator *)animator
{
    NSLog(@"Animator will resume");
}
```

1.5　小结

无论是从开发者的立场还是从 iOS 系统本身未来的发展来看，UIKit Dynamics 都是一个令人感兴趣的话题。苹果公司在花大力气将软件推广到现实世界，希望能够做到让用户同应用的交互就像同真实世界交互的体验一样。用户期望应用对人类指令的反应同周围现实世界的事务一样。对于苹果公司来说这并不新鲜，传统 iPhone 最大的一个卖点就是动量卷动(momentum scrolling)，现在它们又为开发者提供了工具，以实现向程序中添加这些功能。

本章介绍了 UIKit Dynamics 的基本概念和基本组件，不过这些方法真正的强大之处还在于开发者的运用。这个框架有无限的潜能和各种组合，开发者利用这些功能做出的产品可能令苹果公司自己都感到惊讶。用户体验已经被重新定义，那么可以确定的是软件提供真实的物理响应已经不再是可有可无的，用户也非常期待这样的技术。