The Pragmatic Programmers

TURING 图灵程序设计丛书 移动开发系列

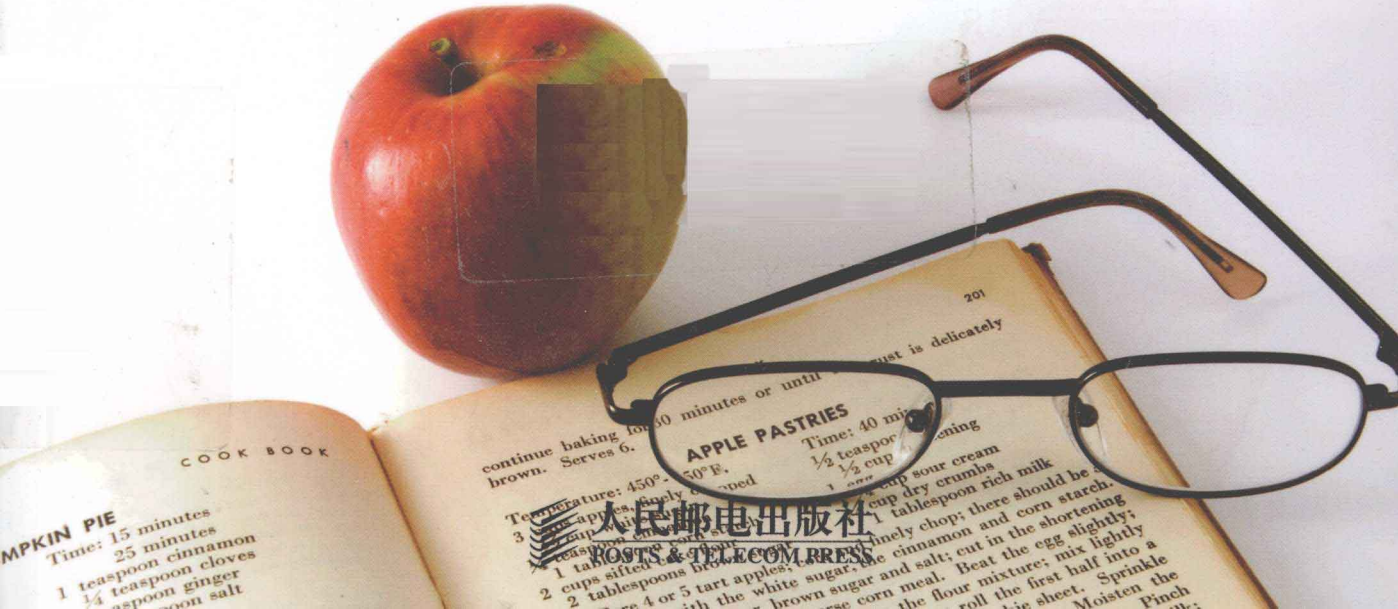
iOS应用开发攻略

iOS Recipes

Tips and Tricks for Awesome iPhone and iPad Apps

- 针对问题提供解决方案
- 代码丰富，实战性强
- 助你iOS开发中攻城拔寨

[美] Matt Drance Paul Warren 著
刘威 译



人民邮电出版社
POSTS & TELECOM PRESS

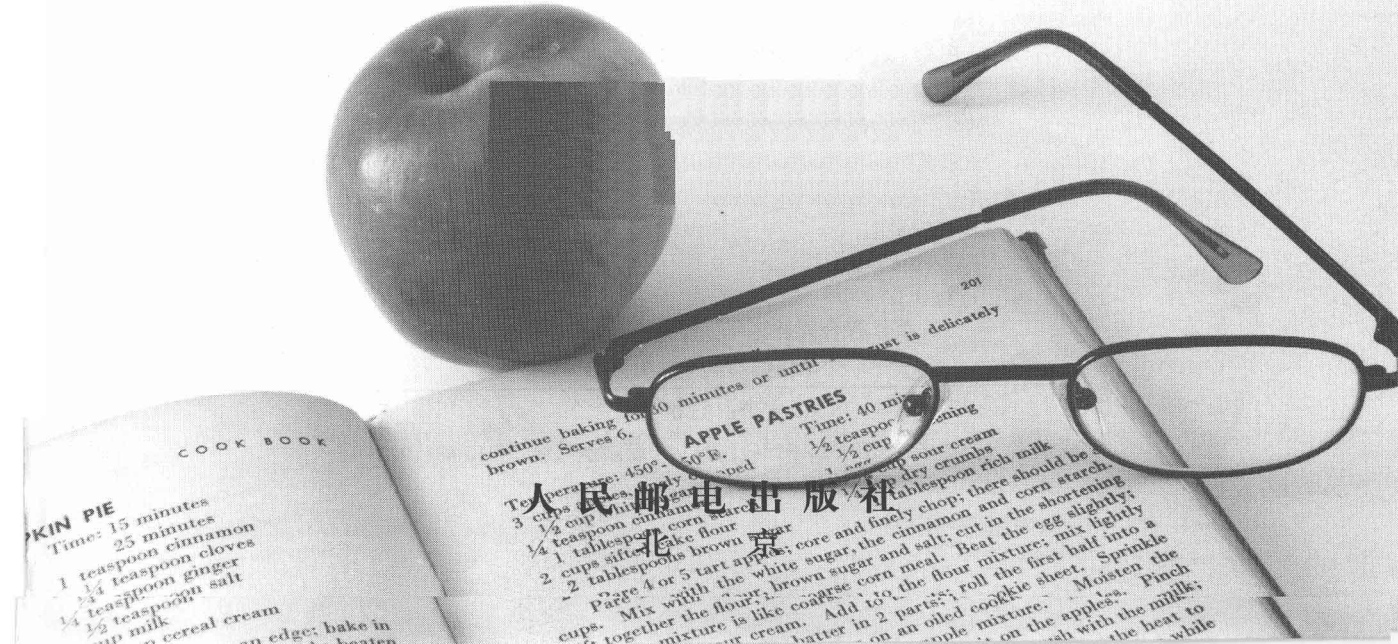
TURING 图灵程序设计丛书 移动开发系列

iOS应用开发攻略

iOS Recipes

Tips and Tricks for Awesome iPhone and iPad Apps

[美] Matt Drance Paul Warren 著
刘威 译



图书在版编目（CIP）数据

iOS应用开发攻略 / (美) 德兰斯 (Drance, M.), (美) 沃伦 (Warren, P.) 著 ; 刘威译. -- 北京 : 人民邮电出版社, 2012.9
(图灵程序设计丛书)
书名原文: iOS Recipes:Tips and Tricks for Awesome iPhone and iPad Apps
ISBN 978-7-115-29178-3

Ⅰ. ①i… Ⅱ. ①德… ②沃… ③刘… Ⅲ. ①移动电话机一游戏程序一程序设计 Ⅳ. ①TN929.53②TP311.5

中国版本图书馆CIP数据核字(2012)第188420号

内 容 提 要

本书收录了最新的iOS软件开发的最佳做法，涵盖了应用开发及构建优雅解决方案的必备知识，包括：编写通用的启动画面和嵌入式Web浏览器；构建复杂表视图；使app或游戏活灵活现的填充、变换和动画；通过手势、转换和自定义控件改善UI；用基本技术知识避免代码重复，解决复杂问题（如上传大文件到Web服务器）等。

本书适合移动开发人员阅读。

图灵程序设计丛书

iOS应用开发攻略

◆ 著 [美] Matt Drance Paul Warren
译 刘 威
责任编辑 傅志红
执行编辑 李 瑛

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：10
字数：236千字 2012年9月第1版
印数：1－4 000册 2012年9月北京第1次印刷

著作权合同登记号 图字：01-2012-4259号

ISBN 978-7-115-29178-3

定价：35.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

本书赞誉

如果只能选择一位老师学习最好的iOS开发方法，我会选择Matt Drance。本书也没让我失望，因为我立刻就用上了几招。我希望尽快用到更多招数，尤其是Paul的好玩的图形与动画技法！

——Brent Simmons，NetNewsWire的开发人员

本书对一系列“他们是怎么做的？”问题做了很好的解答。对于想在自己的苹果应用中点缀亮点的人来说，这是一本必读的书。

——Justin Williams，Second Gear主程序员

无论对初学者还是有经验的开发人员来说，这都是一本很棒的书。书中充满了极为有用的最新示例程序，示范了如何往项目中添加专业级的功能，而且讲解精彩，代码给力。

——Michael Hay，Black Pixel公司高级开发人员

强烈推荐这本书！书中介绍的很多技巧与窍门别处都难以找到。我宁愿从书架上（或iBooks中）取出一本书，来研习曾经看到过的那段代码，书上的代码是不会突然消失的；而去网上找的话，说不定早被网站删掉了。这本书肯定是我要从书架上取出的一本。

——Marcus S. Zarra，Zarra Studios公司所有人

即使你在自己的苹果应用中只用到了书中的一招，那么买这本书也值了。我很快就发现书中马上能用的招数简直太多了。如果你靠写iOS应用来挣钱，而且惜时如金，那么时刻要把书放在触手可及之处，否则你会抓狂的！

——Mike Clark，Clarkware创始人

前　言

对开发者来说，iOS是个神奇的平台。它那不可思议的触摸屏和交互模式，为应用程序打开了全新的篇章。已经有卓越的开发人员开发出了几年前还无法想象的软件。iPhone、iPod Touch和iPad的便携性意味着我们可以将其随身携带，而且其不错的电池待机时间意味着我们可以持续使用。显而易见，iOS指明了未来的方向（我只能对我那台2007年的MacBook Pro装的雪豹操作系统说声抱歉，我开发软件及处理照片时经常会用到它）。显然计算技术已经改变，而且不会再回到2005年的方式。

这令人兴奋。谁不想为这些神奇的设备开发软件呢？

另一方面，实际上我们开始学习如何为iOS及其基于触控的框架开发软件才只有短短的几年时间。当然，读者中有些人一直从事Mac OS X的软件开发，相比大多数从其他平台转而开发iOS的开发人员，你们已领先一步。但是，不要误会。不管你背景如何，在编写iOS的程序时，我们都是处于陌生的领域。尽管我在十几年前就开始编写Cocoa应用程序，并且写了很多有关Mac OS X开发的书和文章，但我在iOS的开发中仍然遇到不少头疼的事，需要在Xcode中埋头查阅其文档。需要弄懂的东西太多，包括如何创建完美的启动画面，如何最有效地用表格与滚动视图实现我们的目标，如何访问新兴的社交应用所使用的众多网络服务，以及如何利用iOS的运行库而不是与之对抗。

很幸运，我们不必自己解决所有这些问题。Matt和Paul（本书的作者）已经在这本攻略书中收集了一系列实例，并且收录了最新的iOS软件开发的最佳做法。结果就是为我们贡献了一系列了不起的、针对目标问题的具体解决方案，我们可以在需要时尽情查阅。

但不止如此。尽管本书由一些完全自成体系的独立章节组成，但读完全书可以让我们对Matt和Paul如何施展其技艺大有领悟。当我阅读本书初稿时，那种感觉就像在观赏我喜爱的厨师现场烹饪美食，我可以向他们学习如何处理手头的工作，甚至包括如何处理我本以为自己已经掌握的简单任务。

所以，且搬把椅子坐下来，跟我最喜爱的两位iOS开发者学上一手！然后，放手去开发几年前你还可望不可及的那些软件吧！

James Duncan Davidson

2011年4月

引　言

作为程序员，我们的目标是解决问题。问题有时困难有时容易，有时又充满乐趣。也许它们算不上我们常说的“问题”，但是需要我们来找到解决办法。

作为作者，我们的目标是帮助读者更好、更快地解决问题（最好是先保证好，然后再保证快）。于是我们决定写这本攻略书，重点讨论一组我们要直接面对的具体任务与问题，而不是泛泛地讨论编程问题。

这不是说这本书不会教大家东西。攻略书的好处在于，能针对读者不易自己找到办法的问题提供可信赖的解决方案。缺点在于，它会让你忍不住把解决方案直接复制粘贴到自己的项目中，而不去花时间理解这些解决方案。能减少编码量从而节省时间总是件好事，但是思考并弄懂时间是如何节省下来的，今后怎样才能节省更多时间，同样非常重要。

如果你熟悉iOS SDK，打算提高app的质量与效率，就请阅读本书。本书不会教你如何编写app，但我们希望帮助你把app开发得更好。如果你是一名高级开发人员，那么可能会发现采用书中的某些复杂技巧能帮你节省时间，避免麻烦。

大部分攻略都是本着保证最大可复用性的原则来编写的。我们并不只是示范可以解决问题的一种技巧或一段代码。相反，我们着眼于创建读者可以集成到自己的iPad或iPhone app中的解决方案。有些方案可能无需改动就能应用到你的项目中，但你尽可以把本书当做一般的“烹饪书”来用。按照食谱烹调时，我们可以根据喜好及需要增减配料。开发自己的app和项目时也是如此：你尽可以扩充或修改本书中各攻略所附的项目，使之满足自己的特定需要。

书中的攻略能从头到尾完整地教你如何完成任务，但是我们希望这些攻略可以激发你去思考在什么情况下、为什么要选择某种方案。通常会有多种选择，尤其是在像Cocoa这样的环境下。有多种选择，就会有多种做法。为保持一致，我们对本书中的某些模式与做法预先做了一些约定。这些技巧中有些你可能比较熟悉，有些技巧的用法你可能没有想到，有些对你来说可能是全新的技巧。不管怎样，我们先对这些约定做个说明，以免读者觉得意外。

格式与语法

本书中有些代码片段为了排版不得不采用特殊的格式。Objective-C这样啰嗦的语言在有字数限制的情况下无法保证效果，所以有时某些代码会显得很特别。读者可能会遇到简短的方法名或变量名、看起来过多的临时变量，还有奇怪的换行。我们力图保留Cocoa惯例的“精神”，但有几

处为了排版只好妥协。如果编码风格偶尔突然改变，请不要奇怪。

范畴

相当一部分攻略为了完成任务而使用了对苹果公司标准类的范畴。范畴是Objective-C编程语言的一项异乎强大的功能，并常让新手Cocoa程序员望而却步。范畴也能够轻易破坏命名空间，在复杂的类层次结构中引入或屏蔽令人意想不到的行为。我们不必害怕它，但要重视它。考虑范畴的时候，需要注意以下几点。

- 想一想是否子类或者新的类会更加合适。如苹果公司的《Objective-C编程语言》手册中所述，“不能用范畴替代子类”。
- 扩展不归自己掌管的类（比如`UIApplication`）时，一定要给范畴方法加前缀，以避免与将来API的名字发生冲突。本书中，新的范畴方法都使用前缀`prp_`。
- 决不能在范畴中重载已有的方法，比如`-drawRect`。这样会屏蔽源类中的实现，从而破坏继承树。

合成的实例变量

我们很少在本书的头文件或例子中声明实例变量。我们全部采用Objective-C 2.0的属性，利用运行库先进的实例变量合成功能来声明类存储。这样做能减少输入与阅读的代码量，让我们可以专注于攻略本身。我们在本书攻略35会作进一步解释。

私有类扩展

私有类扩展是Objective-C的另一项比较新的功能，在本书中经常使用。私有扩展可以减少头文件中的干扰而提高可读性，而且为代码的利用者或维护者提供了更为清晰的描述。在攻略35中，我们向不熟悉私有类扩展或实例变量合成的读者介绍了这两种技术。

`-dealloc`中的清理工作

在`-dealloc`中，除了释放所有相关实例变量之外，我们的例子还把它们设为`nil`。这一做法在Cocoa程序员中争议较大，争论的双方各有道理。本书无意参与这场争论：我们把实例变量设为`nil`，但读者不必非得这样做。如果读者不喜欢`-dealloc`中的`nil`，大可不必写到自己的代码中。

块与委托

块是在Mac OS X雪豹与iOS 4.0中新添加到C和Objective-C中的一项功能。因为这项功能还比较新，所以对于何时使用块、何时使用委托的争论还在继续中。本书中我们在觉得合适的时候两

者同时使用。读者可以随意向使用委托的攻略中添加块，或者反过来也行。我们的最终目的是帮助读者找到最简单、最自然的解决方案。

最重要的是，本书旨在减少读者代码的复杂性与重复。不求速战速决地解决一个问题，而是选择长期有效的解决方案。我们希望本书的这些思想会在iOS开发道路上对你有所帮助。

网上资源

本书有自己的网页，http://pragprog.com/titles/cdirec。在这里读者可以找到有关本书的更多信息，并通过以下方式进行参与：

- 访问书中所有示例程序的完整源代码；
- 参加讨论组，与其他读者、iOS开发者和作者进行交流；
- 报告勘误，包括对内容的建议和提交排印错误，帮助改进本书。

说明：如果你读的是电子版，你可以通过点击代码清单前面的方框，直接下载源文件。

致　　谢

本书的审稿人由全明星阵容组成，感谢他们为此付出的极为宝贵的时间。Colin Barrett、Mike Clark、Michael Hay、Daniel Steinberg、Justin Williams和Marcus Zarra，他们全都非常友好、乐于助人，并积极地帮助我们完善这本书。我们通过电子邮件、Twitter、iChat、午餐会面以及PragProg.com上的讨论组所收到的反馈意见对于本书也至关重要。感谢大家对本书的帮助。

Matt Drance

如果不是喜欢这一主题，就不会写这样一本书。本书的主题来自库比蒂诺（苹果公司全球总部所在地）的上百位才华横溢的热心人士在过去近十年孜孜不倦的努力。我必须感谢我的多位朋友和在苹果公司的前同事，他们创造了这个神奇的平台，感谢这些工程师、产品经理、布道师、技术作者、支持人员，感谢每个人。要不是大家全力以赴，不可能做出iOS这样的产品。

没有苹果公司，就没有写作这本书的机会，但本书的最终面世还要归功于Dave、Andy、Susannah和PragProg的其他职员。当我旁骛于日常工作与其他事务时，我们的编辑Jill Steinberg勇敢而耐心地承担了本书的其他工作。写书一直是我的一个个人目标，这么早就得以实现我非常高兴。感谢大家给我的机会。

但整个过程中，最要感谢的是朋友和家人对我的支持。我所做的一切都是为了我了不起的妻子和儿子。当独立开发者是不错，但还远不能跟当丈夫和爸爸相比。

Paul Warren

我要感谢苹果公司这些了不起的人的工作，感谢他们构建了这个神奇的平台，它是我们每天的游乐场。还要感谢Jill和PragProg.com的团队，感谢他们使本书的创作和出版过程如此顺利。也感谢我们卓越的开发者社区，感谢他们的分享和鼓励。

我美丽的妻子和女儿们，你们对家里的写作新手表现出了极大的耐心，梦里肯定都会萦绕着“你觉得这样如何？”之类的话。此外，你们给了我一个和谐美满的家庭，用你们无尽的爱让我的生活充满欢声笑语，为此我将永远充满感激。

目　录

第1章 UI 攻略

关于UI的攻略，我们轻易就能写出整整一本书。毕竟iOS SDK中值得讨论的类库与模式似乎无穷无尽。最终，我们决定专注于读者会反复遇到、却又记不清以前的解决方式的那些简单的模式与问题，展示针对这些问题的优秀解决方案。

这一章，我们将介绍有关视图切换、网页内容、触摸处理和定制控件的攻略。这些攻略可供读者拿来使用，也可能纯粹为了启发读者去思考如何让自己的代码可以复用于将来的项目。

攻略1 添加基本的启动画面切换

问题

应用程序启动时，从默认图像到实际UI的切换如果很生硬，带给用户的第一印象会很糟糕。我们想让从应用程序的启动图像到初始UI的切换尽可能平滑流畅，但不清楚如何用最简洁的方式实现。

解决方案

iOS app启动的直观体验是这样的：

(1) 用户点击app图标；

(2) app的默认图像逐渐放大显示到屏幕上；

(3) app的初始UI加载到内存；

(4) UI显示到屏幕上，取代默认图像。

如果默认图像是品牌的横幅或其他特有的图片，那么由它到实际UI的切换可能会让用户觉得生硬。我们需要一种从启动画面到运行的程序之间的平滑切换。这可以通过许多方式来实现，我们从最简单的一种方式讲起，这应该是一种普遍适用的方式。我们先来处理一个纵向的iPhone程序，接下来看看支持各种方向的iPad版本。初始画面如图1所示。

图1　启动画面与初始UI

最简单的“启动画面切换”是默认图像淡出，而UI同时淡入。这种切换容易实现，成本不高，又能让用户体验截然不同。想想看，这是用户第一眼看到的东西，没有理由不做得平滑顺畅。

为了使默认图像淡出屏幕，首先需要显示一个视图，由它显示这幅图像，然后把这个视图淡出。这很容易，先创建一个可用于任何项目的简单的视图控制器，由它使用定制的启动屏幕图像，并定义一个执行淡出的-hide方法。

BasicSplashScreen/PRPSplashScreen.h

```
@interface PRPSplashScreen : UIViewController {}

@property (nonatomic, retain) UIImage *splashImage;
@property (nonatomic, assign) BOOL showsStatusBarOnDismissal;
@property (nonatomic, assign) IBOutlet id<PRPSplashScreenDelegate> delegate;

- (void)hide;

@end
```

接口中有一个delegate属性，声明为id <PRPSplashScreenDelegate>类型。这个PRPSplashScreenDelegate协议定义在独立的头文件中，向相关程序传达启动画面的状态：启动画面何时开始显示、何时开始切换及何时结束切换。

读者肯定在很多地方做过委托所做的事情，但很可能以前没有定义过委托。我们来看看这个协议的定义，请注意@optional关键字，这意味着该委托不要求实现它声明的所有方法。一个对象如果想知道启动画面的状态，可以声明自己遵守PRPSplashScreenDelegate协议，实现一个或多个委托方法，并把自己赋值给启动画面的delegate属性。

BasicSplashScreen/PRPSplashScreenDelegate.h

```
@protocol PRPSplashScreenDelegate <NSObject>

@optional
- (void)splashScreenDidAppear:(PRPSplashScreen *)splashScreen;
- (void)splashScreenWillDisappear:(PRPSplashScreen *)splashScreen;
- (void)splashScreenDidDisappear:(PRPSplashScreen *)splashScreen;

@end
```

PRPSplashScreen在-loadView方法中构建其视图，这样就不必在每次用到它时都去拖XIB文件了，因而也更便于我们把它应用到项目中。view属性设置为一个以居中的图像充满屏幕的图像视图。

BasicSplashScreen/PRPSplashScreen.m

```
- (void)loadView {
    UIImageView *iv = [[UIImageView alloc] initWithImage:self.splashImage];
    iv.autoresizingMask = UIViewAutoresizingFlexibleWidth |
        UIViewAutoresizingFlexibleHeight;
    iv.contentMode = UIViewContentModeCenter;
    self.view = iv;
    [iv release];
}
```

现在来看看splashImage属性，它是可写的，所以必要时我们可以设置定制的切换图像。但是我们这里只使用Default.png作为启动图像，因为这个攻略的要点是创建平滑流畅的切换。所以，我们写了一个默认加载Default.png的懒初始化方法。如果是从默认的图像做切换，就无需改动这个属性。我们使用+[UIImage imageNamed:]来保证使用适当比例的图像（比如对于Retina显示屏使用Default@2x.png）。

BasicSplashScreen/PRPSplashScreen.m

```
- (UIImage *)splashImage {
    if (splashImage == nil) {
        self.splashImage = [UIImage imageNamed:@"Default.png"];
    }
    return splashImage;
}
```

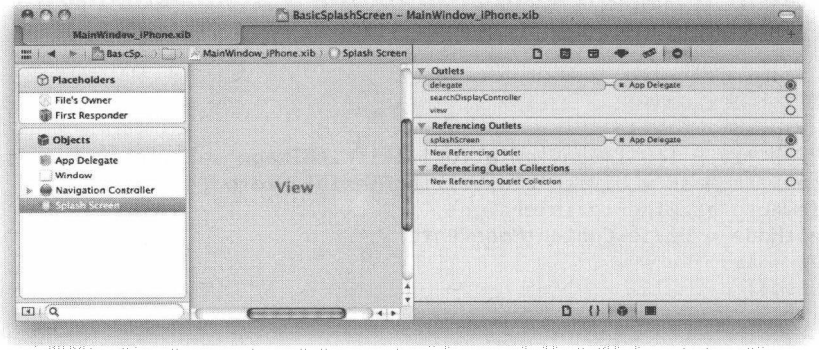
设置启动画面很简单，只是从app的根视图控制器显示一个模态视图控制器而已。我们将在程序启动时，在添加了根视图之后、显示主窗口之前做这件事。时机很重要：根视图控制器在自身的视图没有准备好之前，不会正常显示模态视图控制器。本章的BasicSplashScreen项目中，我们在代码中也设定了一种溶解风格（淡入淡出）的切换。因为启动画面默认使用启动图像，所以不需要我们自己指定。

BasicSplashScreen/iPhone/AppDelegate_iPhone.m

```
- (BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [self.window addSubview:self.navController.view];
```

```
    self.splashScreen.showsStatusBarOnDismissal = YES;
    self.splashScreen.modalTransitionStyle = UIModalTransitionStyleCrossDissolve;
    [self.navController presentModalViewController:splashScreen animated:NO];
    [self.window makeKeyAndVisible];
    return YES;
}
```

如果打开MainWindow_iPhone.xib，我们会看到XIB文件中定义了一个PRPSplashScreen对象，如图2所示。在Interface Builder中，这个对象连接到app委托的splashScreen属性。前面的代码引用了这个属性，以显示启动画面。



启动画面在各自的MainWindow XIB文件中作初始化，并连接到应用程序委托的splashScreen属性。app委托也连接到启动画面的delegate属性。

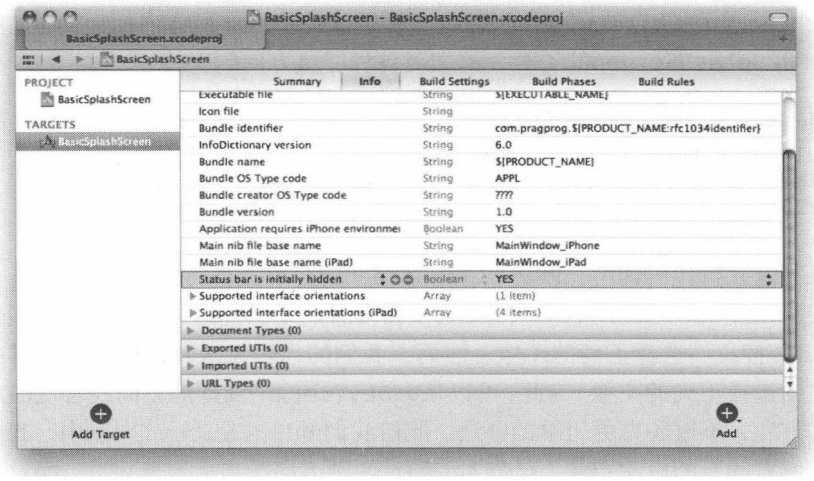
图2　在Interface Builder中连接启动画面

窗口可见之后，启动画面视图控制器会收到标准的UIViewController消息，包括-viewDidAppear:。它意味着要开始画面切换，而且实现起来非常简单。我们首先通知委托，告诉它启动画面视图已经显示了，以便它在必要时为画面切换做准备。首先检查委托是否实现了适当的方法很重要，因为在委托协议中这些方法声明为可选的。向委托发送消息之后，我们发送-hide消息来执行启动画面切换。请注意此处我们使用了performSelector:withObject:afterDelay:，这样可以让UIKit运行循环（run loop）完成viewDidAppear机制。在自身的viewWillAppear:或viewDidAppear:方法中解除（dismiss）视图控制器会导致系统混乱，每个动作需要分离开来，并相互独立。

BasicSplashScreen/PRPSplashScreen.m

```
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
    SEL didAppearSelector = @selector(splashScreenDidAppear:);
    if ([self.delegate respondsToSelector:didAppearSelector]) {
        [self.delegate splashScreenDidAppear:self];
    }
    [self performSelector:@selector(hide) withObject:nil afterDelay:0];
}
```

-hide方法首先检查是否要在淡出时显示状态条，然后使用标准的-dismissModalViewControllerAnimated:方法来执行画面切换。添加这个处理是为了应对启动时不显示状态条而在UI中却显示状态条的情形。要实现这种效果，需在app的Info.plist文件中把UIStatusBarHidden设为YES，把启动画面的showsStatusBarOnDismissal属性设为YES。启动画面会负责重新激活状态条，所以我们不必自己在委托方法中来做（如图3所示）。



把UIStatusBarHidden键设为YES以便在启动时隐藏状态条。如果要在主UI中显示状态条，需要把启动画面的showStatusBarOnDismissal属性设为YES。

图3　启动时隐藏状态条

BasicSplashScreen/PRPSplashScreen.m

```
- (void)hide {
    if (self.showsStatusBarOnDismissal) {
        UIApplication *app = [UIApplication sharedApplication];
        [app setStatusBarHidden:NO withAnimation:UIStatusBarAnimationFade];
    }
    [self dismissModalViewControllerAnimated:YES];
}
```

启动画面也会通过转发标准的视图控制器方法-viewWillDisappear:和-viewDidDisappear:，把画面切换的进度通知给委托。app委托使用相应的-splashScreenDidDisappear:委托方法来删除不再需要的启动画面。

BasicSplashScreen/iPhone/AppDelegate_iPhone.m

```
- (void)splashScreenDidDisappear:(PRPSplashScreen *)splashScreen {
    self.splashScreen = nil;
}
```

以iPhone为目标环境运行BasicSplashScreen项目，看看从启动画面到UI的画面切换。委托的连接设置是在MainWindow_iPhone.xib和MainWindow_iPad.xib中做的，所以在代码中不必访问delegate属性。我们用来展示本书细节页面的PRPWebViewController类，会在攻略6中再详细讲解。

这个解决方案现在只能执行竖屏的画面切换，对于多数iPhone app来说就够了。而iPad app会经常以横屏和竖屏两种方式运行。因为UIViewController提供了自动旋转行为，而PRPSplashScreen继承了UIViewController，所以支持多种屏幕方向相当简单。我们先创建一个iPad专用的PRPSplashScreen的子类，添加对各种屏幕方向的支持。

BasicSplashScreen/iPad/PRPSplashScreen_iPad.m

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
        (UIInterfaceOrientation)toInterfaceOrientation {
    return YES;
}
```

子类中只写这几行代码即可，PRPSplashScreen的所有其他行为都原封不动。

最后一件事是提供一幅新的启动图像。当支持多种启动方向时，要提供默认图像的竖屏和横屏版本，UIKit会替我们选择正确的图像。然而，我们的代码无法知道使用的是哪幅，因此不能为启动画面视图选择正确的图像。为此，可以从UIDevice检测设备的方向或者从UIApplication检测状态条的方向，不过还有更简单的办法。既然我们的目的是让LOGO居中，我们可以制作一幅1024×1024像素的新的启动图像。这个大小满足两种方向的最大屏幕尺寸，而且无论设备如何旋转都能够在充满屏幕的同时保持居中。即使动态旋转发生在画面切换之前，图像仍可以保持居中。我们把这幅图像加到app之中，并通过由PRPSplashScreen定义的splashImage属性把它设置为启动画面的启动图像。

BasicSplashScreen/iPad/AppDelegate_iPad.m

```
- (BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [self.window addSubview:self.splitViewController.view];

    UIImage *splash = [UIImage imageNamed:@"splash_background_ipad.png"];
    self.splashScreen.splashImage = splash;

    self.splashScreen.showsStatusBarOnDismissal = YES;
    self.splashScreen.modalTransitionStyle = UIModalTransitionStyleCrossDissolve;
    [self.splitViewController presentModalViewController:splashScreen animated:NO];

    [self.window makeKeyAndVisible];

    return YES;
}
```

初始化代码的其余部分跟iPhone版本相同。请运行iPad版的BasicSplashScreen，观察横屏与竖屏状态下平滑流畅的画面切换，如图4所示。现在我们创建了一个从有特色的默认图像到app的初始UI之间的基本画面切换，这个效果容易复用，既能用于iPhone，又能用于iPad。