iOS 5 Programming Cookbook

第十二章
多任务

版本 1.0

翻译时间：2012-07-30

DevDiv 翻译： kyelup cloudhsu 耐心摩卡
wangli2003j3 xiebaochun dymx101
Jimmylianf BeyondVincent

DevDiv 校对： laigb kyelup

DevDiv 编辑： BeyondVincent

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问 DevDiv 移动开发论坛 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

关于本文的翻译

感谢 kyelup、cloudhsu、耐心摩卡、wangli2003j3、xiebaochun、dymx101、jimmylianf 和 BeyondVincent 对本文的翻译，同时非常感谢 laigb 和 kyelup 在百忙中抽出时间对翻译初稿的认真校验，指出了文章中的错误。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

读者查看下面的帖子可以持续关注章节翻译更新情况

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译各章节汇总

更多学习资料

我们也为大家准备了 iOS6 新特征的相关内容，请参考下面的帖子：

iOS6 新特征：参考资料和示例汇总------持续更新中

目录

前言

参考帖子

[DevDiv翻译]iOS 5 Programming Cookbook翻译_前言

第 1 章 基础入门

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第一章_基础入门

第 2 章 使用控制器和视图

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(上)

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(下)

第 3 章 构造和使用 Table View

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第三章_构造和使用 Table View

第 4 章 Storyboards

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第四章_Storyboards

第 5 章　并发

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第五章_并发

第 6 章 定位核心与地图

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第六章_核心定位与地图

第 7 章 实现手势识别的功能

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第七章_实现手势识别

第 8 章 网络，JSON，XML 以及 Twitter

参考帖子

[DEVDIV 翻译] iOS 5 Programming Cookbook 翻译_第八章_网络，JSON，XML 以及 Twitter

第 9 章 音频和视频

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第九章_音频和视频

第 10 章 通讯录

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十章_通讯录

第 11 章 照相机和图片库

参考帖子

[DEVDIV 翻译]iOS 5 Programming Cookbook 中文翻译_第十一章_照相机和图片库

第 12 章 多任务

12.0 概述

多任务允许“后台执行”（background execution），意味着程序可以像往常一样工作---运行任务，产生大量的新线程，等候通知并对事件作出反应---但不会在屏幕上显示任何东西或者以任何方式和用户交互。当用户按下设备上的 Home 按钮，这在之前的版本将终止 iPhone 或者 iPad 上的应用程序，现在程序会被送到后台。

在支持多任务的 iOS 版本上运行的应用程序默认选择在后台执行。如果你将程序更新到 iOS SDK 4.0 及后续版本，你可以选择不在后台执行，如你在 12.10 小节所见。如果你这样做，你的程序将在用户按下 Home 按钮时退出，像以前一样。

当我们的程序转到后台（比如用户按下了 Home 按钮）然后回到前台（当用户再次选择程序时），系统会发送一些不同的消息，期待被我们指派的应用程序委托对象接收。比如，当我们的程序被送到后台，应用程序委托将会收到 applicationDidEnter Background: 方法，当程序被用户切到前台，应用程序委托将收到 applicationWillEnterForeground:委托消息。

除了这些委托消息，在程序前后台切换时，iOS 也向运行中的程序发送通知。程序转入后台时发送的通知是 UIApplicationDidEnterBackgroundNotification，从后台切回前台发送的通知为 UIApplicationWillEnterForegroundNotification。你可以使用默认通知中心来注册这些通知。

12.1. 检测多任务的可用性

12.1.1. 问题

你想知道正运行你程序的 iOS 设备是否支持多任务。

12.1.2. 方案

调用 UIDevice 对象的 isMultitaskingSupported，如下：

```
- (BOOL) isMultitaskingSupported{
BOOL result = NO;
if ([[UIDevice currentDevice]
respondsToSelector:@selector(isMultitaskingSupported)]){ result = [[UIDevice currentDevice] isMultitaskingSupported];
}
return result;
}
```

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
if ([self isMultitaskingSupported]){
NSLog(@"Multitasking is supported.");
} else {
NSLog(@"Multitasking is not supported."); }
self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor]; [self.window makeKeyAndVisible];
return YES;
}
```

12.1.3. 讨论

你的程序，根据它的目标 iOS 设备类型，可以运行在不同操作系统的一些不同的设备上。比如，你用 iOS SDK 5.0 编译你的程序并设置部署目标 OS 为 4.0，你的应用程序就可以运行在 iPhone 3G, iPhone3GS, iPhone4 和 iPod Touch（2 代或 3 代）之上。此外，设备可能安装了 iOS 5.0 或以上版本，但是基本硬件可能不能强大到能支持多任务。因此，你的程序必须注意，在尝试成为一个多任务程序之前，要了解在该设备及其 iOS 版本是否支持多任务。

12.1.4. 参考

12.2. 在后台完成一个长期任务（Long-Running Task）

12.2.1. 问题

你希望能在程序被送入后台时，向 iOS 借点时间，来完成一个长期任务。

12.2.2. 方案

使用 UIApplication 的 beginBackgroundTaskWithExpirationHandler: 实例方法。在你完成任务后，调用 UIApplication 的 endBackgroundTask:方法。

12.2.3. 讨论

当一个 iOS 应用被送到后台，它的主线程会被暂停。你用 NSThread 的 detachNewThreadSelector:toTar get:withObject:类方法创建的线程也被挂起了。如果你想在后台完成一个长期任务，就必须调用 UIApplication 的 beginBackgroundTaskWithExpirationHandler:实例方法，来向 iOS 借点时间。UIApplication 的 backgroundTimeRemaining 属性包含了程序完成他的任务可以使用的秒数。如果在这个期限内，长期任务没有被完成，iOS 将终止程序。每个对 beginBackgroundTaskWithExpirationHandler:方法的调用，必须要相应的调用 endBackgroundTask:方法（UIApplication 的另一个实例方法）。也就是说，如果你向 iOS 要更多时间来完成一个任务，你必须告诉 iOS 你什么时候能完成那个任务，那时，你的程序将

和其所有被暂停的线程被放入后台。

当你的程序在前台时，UIApplication 的 backgroundTimeRemaining 属性等于 DBL_MAX 常量，这是 double 类型可表示的最大值（和这个值相当的 integer 通常等于-1）。在 iOS 被要求在程序被完全挂起之前给于更多的执行时间，这个属性指明了在完成任务前程序拥有多少秒。

在程序中你可以多次调用 beginBackgroundTaskWithExpirationHandler:方法。要记住的重点是，当 iOS 为你的程序返回一个 token 或者任务标识（task identifier）时，你都必须调用 endBackgroundTask:方法，在运行的任务结束时，用来标志任务结束。如果你不这么做的话，iOS 会终止你的程序。

在后台时，程序不应该执行完全的功能，也不应该处理大量数据。事实上，他们只应该完成一个长期任务。

比如，一个程序正在调用一个 web service API，并且还没有从服务器上的那个 API 接收到响应。在此期间，如果程序被送入后台，它可以请求更多的时间，直到它从服务器收到响应。一旦响应被接收，程序必须保存其状态，并调用 UIApplication 的 endBackgroundTask:实例方法将任务标记为完成。

让我们看一个例子。我将从在应用程序委托中定义一个 UIBackgroundTaskIdentifier 类型的属性开始。同时，让我们定义一个 NSTimer，当程序被送到后台时，我们将用它每隔 1 秒向控制台窗口输出一条消息：

```
#import <UIKit/UIKit.h>
@interface Completing_a_Long_Running_Task_in_the_BackgroundAppDelegate : UIResponder <UIApplicationDelegate>
@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, unsafe_unretained) UIBackgroundTaskIdentifier backgroundTaskIdentifier;
@property (nonatomic, strong) NSTimer *myTimer;
@end
```

接下来我们继续同步属性：

```
#import "Completing_a_Long_Running_Task_in_the_BackgroundAppDelegate.h" @implementation
Completing_a_Long_Running_Task_in_the_BackgroundAppDelegate
@synthesize window = _window;
@synthesize backgroundTaskIdentifier; @synthesize myTimer;
```

现在，让我们创建定时器，并在程序被送到后台时启动它：

```
- (BOOL) isMultitaskingSupported{
BOOL result = NO;
if ([[UIDevice currentDevice]
respondsToSelector:@selector(isMultitaskingSupported)]){ result = [[UIDevice currentDevice] isMultitaskingSupported];
}
return result;
}

- (void) timerMethod:(NSTimer *)paramSender{
NSTimeInterval backgroundTimeRemaining =
[[UIApplication sharedApplication] backgroundTimeRemaining];
if (backgroundTimeRemaining == DBL_MAX){ NSLog(@"Background Time Remaining = Undetermined");
} else {
```

```
NSLog(@"Background Time Remaining = %.02f Seconds",
backgroundTimeRemaining);
} }
- (void)applicationDidEnterBackground:(UIApplication *)application{
if ([self isMultitaskingSupported] == NO){
return; }
self.myTimer =
[NSTimer scheduledTimerWithTimeInterval:1.0f
target:self
selector:@selector(timerMethod:) userInfo:nil
repeats:YES];
self.backgroundTaskIdentifier =
[application beginBackgroundTaskWithExpirationHandler:^(void) { [self endBackgroundTask];
}]; }
```

你可以看到，在后台任务的完成处理者（completion handler）中，我们调用了应用程序委托的 endBackgroundTask 方法。这是一个我们编写的方法，如下：

```
- (void) endBackgroundTask{
dispatch_queue_t mainQueue = dispatch_get_main_queue();
__weak Completing_a_Long_Running_Task_in_the_BackgroundAppDelegate *weakSelf = self;
dispatch_async(mainQueue, ^(void) {
Completing_a_Long_Running_Task_in_the_BackgroundAppDelegate *strongSelf = weakSelf;
if (strongSelf != nil){
[strongSelf.myTimer invalidate];
[[UIApplication sharedApplication] endBackgroundTask:self.backgroundTaskIdentifier];
strongSelf.backgroundTaskIdentifier = UIBackgroundTaskInvalid; }
}); }
```

在长期任务结束后，我们需要做一些事情进行清理：

1. 结束所有的线程和定时器，不管他们是基础定时器还是 GCD 中创建的。
2.调用 UIApplication 的 endBackgroundTask:方法来结束后台任务。
3.将任务标识设置为 UIBackgroundTaskInvalid，标志我们的任务结束。

最后，当我们的应用回到前台，如果我们的后台任务还在执行中，我们需要确保我们在干掉它：

```
- (void)applicationWillEnterForeground:(UIApplication *)application{
if (self.backgroundTaskIdentifier != UIBackgroundTaskInvalid){
[self endBackgroundTask]; }
}
```

在我们的例子中，不论何时程序被送到后台，我们都会要求更多时间以完成一个长期任务（例如，在这里是我们计时器的代码）。在我们的时间里，我们不断的读取 UIApplication 实例中 backgroundTimeRemaining 属性的值，将它打印到控制台。在 UIApplication 的 beginBackgroundTask WithExpirationHandler: 实例方法中，在程序的额外时间内完成一个长期任务之前，我们提供的代码将被执行（一版大概在任务过期前 5 到 10 秒）。在此，我们只要调用 UIApplication 的 endBackgroundTask:实例方法来结束任务。

当一个程序被送到后台而且它从 iOS 请求了额外执行时间，在执行时间完成之前，程序

可以被用户唤醒并被带回到前台。如果你在程序被送到后台时请求过执行一个长期任务，那么此时你必须使用 UIApplication 的 BackgroundTask:实例方法结束这个长期任务。

12.2.4. 参考

12.1 小节

12.3. 在后台接收本地通知

12.3.1. 问题

你希望即使在程序没有运行时，可以向用户展示一个警告。
你希望在没有使用推送消息的情况下，可在程序中创建本地警告。

12.3.2. 方案

实例化一个 UILocalNotification 对象，然后使用 UIApplication 的 scheduleLocalNotification:实例方法安排它：

```
- (BOOL) localNotificationWithMessage:(NSString *)paramMessage actionButtonTitle:(NSString *)paramActionButtonTitle
launchImage:(NSString *)paramLaunchImage applicationBadge:(NSInteger)paramApplicationBadge
secondsFromNow:(NSTimeInterval)paramSecondsFromNow userInfo:(NSDictionary *)paramUserInfo{
if ([paramMessage length] == 0){
return NO; }
UILocalNotification *notification = [[UILocalNotification alloc] init]; notification.alertBody = paramMessage;
notification.alertAction = paramActionButtonTitle;
if ([paramActionButtonTitle length]> 0){
/* Make sure we have the action button for the user to press
to open our application */ notification.hasAction = YES;
} else {
notification.hasAction = NO; }
/* Here you have a chance to change the launch image of your application
when the notification's action is viewed by the user */ notification.alertLaunchImage = paramLaunchImage;
/* Change the badge number of the application once the notification is presented to the user. Even if the user dismisses the
notification, the badge number of the application will change */
notification.applicationIconBadgeNumber = paramApplicationBadge;
/* This dictionary will get passed to your application
later if and when the user decides to view this notification */
notification.userInfo = paramUserInfo;
/* We need to get the system time zone so that the alert view will adjust its fire date if the user's time zone changes */
NSTimeZone *timeZone = [NSTimeZone systemTimeZone]; notification.timeZone = timeZone;
/* Schedule the delivery of this notification 10 seconds from now */ NSDate *today = [NSDate date];
NSDate *fireDate = [today dateByAddingTimeInterval:paramSecondsFromNow]; NSCalendar *calendar = [NSCalendar
autoupdatingCurrentCalendar];
NSUInteger dateComponents = NSYearCalendarUnit | NSMonthCalendarUnit | NSDayCalendarUnit | NSHourCalendarUnit |
NSMinuteCalendarUnit | NSSecondCalendarUnit;
NSDateComponents *components = [calendar components:dateComponents fromDate:fireDate];
/* Here you have a chance to change these components. That's why we retrieved the components of the date in the first place. */
```

```
fireDate = [calendar dateFromComponents:components];
/* Finally set the schedule date for this notification */ notification.fireDate = fireDate;
[[UIApplication sharedApplication] cancelAllLocalNotifications]; [[UIApplication sharedApplication]
scheduleLocalNotification:notification]; return YES;
}
```

12.3.3. 讨论

本地通知是一个警告视图（UIAlertView 类型的对象），在你的程序运行于后台时或者根本没运行的情况下展现给用户。你可以使用 UIApplication 的 scheduleLocalNotification: 实例方法安排本地通知的发送，cancelAllLocalNotifications 实例方法取消所有进行中的本地通知的发送。

你也可以要求 iOS 将来即使在程序没有运行时向用户发送本地通知。这些通知可以是重复的--比如，每周的某个时间。但是，在你指定通知的触发日期时必须特别小心。

举例来说，假设现在是伦敦时间 13:00，现在的时区是 GMT+0，你的程序正运行在用户的设备上。你想在 14:00 向用户发送通知，即使那时程序已退出。现在你的用户在伦敦盖特维克机场飞往斯德哥尔摩的飞机上，斯德哥尔摩的时区是 GMT+1.如果飞行需要 30 分钟，用户到达斯德哥尔摩的时间将是 GMT+0 时区 13:00。但是，当它到达时，iOS 设备会侦测到时区的变化，然后将用户设备的时间调到了 14：30。你的通知本应该在 14：00（GMT+0）时发生，所以当时区变化时，iOS 设备检测到通知应该被显示了（晚了 30 分钟，实际上是新时区），然后显示该通知。

问题是你的通知本来应该在 14:00 GTM+0 或者 15:00 GTM+1 显示，而不是 14:30 GTM+1。为了处理这类情况（由于现代的旅行习惯，它发生几率比你认为的更频繁），当你为通知指定了触发日期和时间，你还应该为这个时间指定时区。

前面的代码并不包括警告视图，你需要编写它来向用户显示些东西。我们继续向程序添加代码，看看在不同情景下面 iPhone 模拟器会发生什么。下面是我们应用程序委托的.h 文件：

```
#import <UIKit/UIKit.h>
@interface Receiving_Local_Notifications_in_the_BackgroundAppDelegate : UIResponder <UIApplicationDelegate>
@property (nonatomic, strong) UIWindow *window;
@end
```

非常棒！现在我们要到 application:didReceiveLocalNotification:方法中，看看是否有本地通知唤醒了我们的应用。如果没有的话，我们安排一个：

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor]; [self.window makeKeyAndVisible];
id scheduledLocalNotification = [launchOptions valueForKey:
UIApplicationLaunchOptionsLocalNotificationKey]; if (scheduledLocalNotification != nil){
/* We received a local notification while
our application wasn't running. You can now typecase the ScheduledLocalNotification variable to UILocalNotification and use it in
your application */
NSString *message = @"Local Notification Woke Us Up"; [[[UIAlertView alloc] initWithTitle:@"Notification"
message:message delegate:nil
cancelButtonTitle:@"OK" otherButtonTitles:nil, nil] show];
NSString *message =@"A new instant message is available. \ Would you like to read this message?";
```

```
/* If a local notification didn't start our application, then we start a new local notification */
[self localNotificationWithMessage:message actionButtonTitle:@"Yes"
launchImage:nil applicationBadge:1
secondsFromNow:10.0f userInfo:nil];
message = @"A new Local Notification is set up \ to be displayed 10 seconds from now";
[[[UIAlertView alloc] initWithTitle:@"Set Up" message:message
delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil, nil] show];
}
return YES; }
```

最后强调一点，我们会在程序中处理 UIApplicationDelegate 的 application:didReceiveLocalNotification:方法，来确保用户打开应用时，因为收到本地通知而向她显示了一条消息：

```
- (void) application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification *)notification{
NSString *message = @"The Local Notification is delivered.";
[[[UIAlertView alloc] initWithTitle:@"Local Notification" message:message
delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil, nil] show];
}
```

现在让我们测试上面的代码。我们的第一种情形是：用户刚刚安装了我们的程序，第一次运行它。图 12-1 显式了我们的所见。

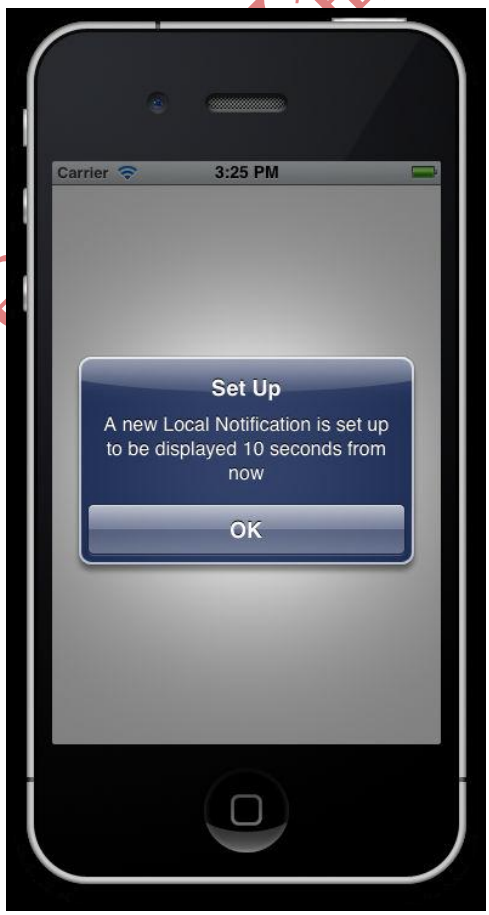

图 12-1. 本地通知被设置的提示

用户点击 OK 按钮并留在程序中。图 12-2 显示了在消息被发送到程序后用户看到的消息。

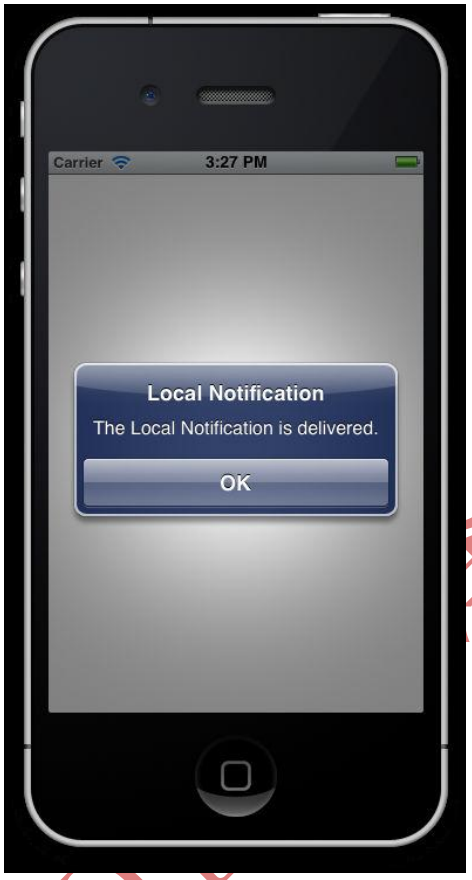

图 12-2. 程序运行时，本地消息被发送。

程序在运行或者即便在后台时（还没有退出），iOS 会调用应用程序委托的 application:didReceiveLocalNotification:方法，让应用程序知道本地消息已经发给我们了。如果用户在程序中，iOS 将不会做任何事情，并且也不会自动显示消息。但是，当程序在后台运行时，iOS 确实会自动显示通知消息。

第二种情形，用户第一次打开我们的程序，就像图 21-1 所示，在按下 OK 按钮之后，马上按下了 iOS 设备上的 Home 按钮，将程序送到后台。现在，当通知被发送，用户会看到类似图 12-3 的一条消息。

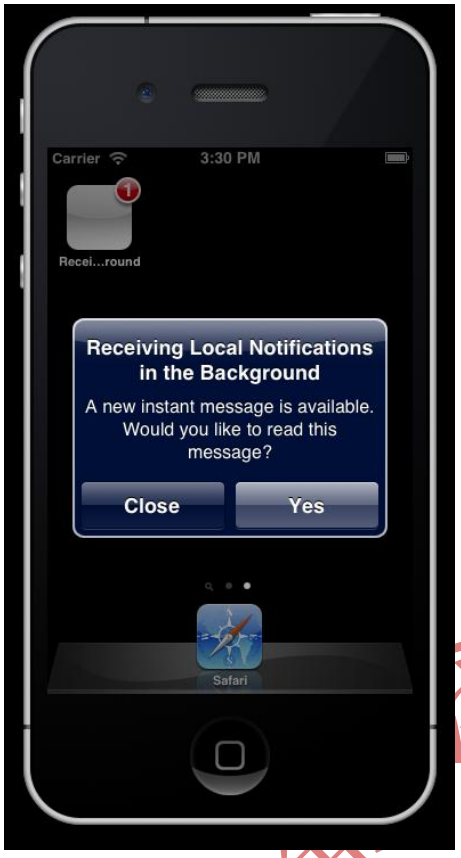

图 12-3. 发送给后台应用的本地通知

因为我们在创建通知时，将本地通知的程序徽章数（application's badge number）设置为 1，我们的程序徽章数在收到通知后马上变为了 1。用户不需要接受或关闭通知，改变徽章数。现在如果用户按下 Yes 按钮，iOS 会运行和本地通知相关的程序，用户会看到如图 12-2 所示的屏幕。请注意在这种情况下，我们的程序没有被关闭，而是被送到了后台。

第三种情形是，当我们的程序第一次运行，就像图 12-1 那样，并且用户将程序送入后台。然后用户双击 Home 按钮，长按程序图标然后会出现关闭程序按钮，以这种方式用户手动的终止了程序，如图 12-4 所示。

图 12-4. 在本地通知被发送前，用户尝试终止程序。

一旦我们的程序终止，本地通知将在数秒后（将通知加入调度后的第 10 秒）显示。当通知被送达，用户会看到类似于图 12-3 所示的屏幕。在用户点击了 Yes 按钮后，iOS 再次启动我们的程序，用户将看到如图 12-5 所示的屏幕。

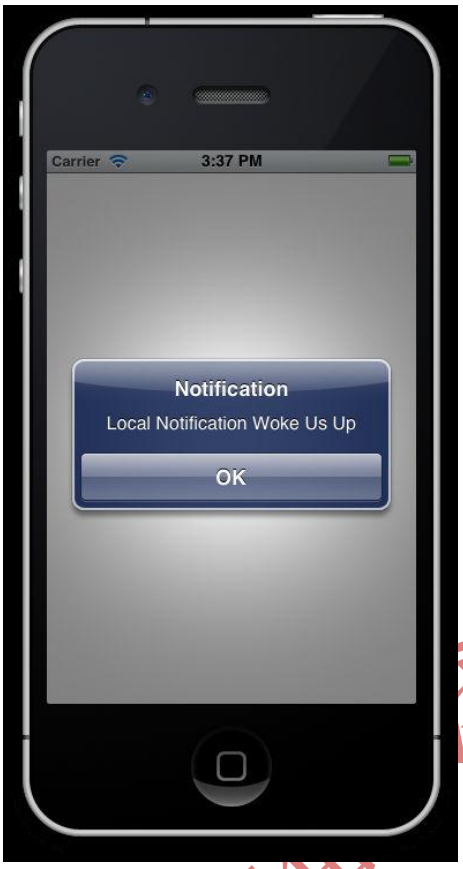

图 12-5. 本地通知唤醒了已终止的程序

所以，你可以直观的看到本地通知如何工作。当我们的程序在前台或后台运行时，iOS 会通过 application:didReceiveLocalNotification:委托方法发送本地通知。但是如果程序是被用户或者 iOS 所终止，我们将通过 didFinishLaunchingWithOptions: 方法收到本地通知（也就是，用户决定查看通知）。我们可以在 didFinishLaunchingWith Options 的参数中使用 UIApplicationLaunchOptionsLocalNotificationKey 键来获得通知。

一个本地通知不必是一个动作通知（action notification）。动作通知有两个按钮。你可以通过 UILocalNotification 的 alertAction 属性，改变其中一个按钮的标题。另一个是 OK 按钮，用于取消警告；你不能改变它的标题或行为。如果一个通知不是动作通知（当 UILocalNotification 的 hasAction 设为 NO 时），通知将仅显式 OK 按钮，点击按钮也不会再次启动你的程序。

12.3.4. 参考

12.4. 在后台播放声音

12.4.1. 问题

你在写一个播放声音文件的程序（比如一个音乐播放器），你希望即使程序在后台运行时，也能播放声音文件。

12.4.2. 方案

在你程序的主.plist（main .plist）创建一个新的 array 键。设置键的名字为 UIBackgroundModes。将 audio 赋值给这个新的键。下面是一个.plist 文件内容的例子，加入了前面提到的键和值：

```
<dict> ...
...
... <key>UIBackgroundModes</key> <array>
<string>audio</string> </array>
... ...
... </dict>
```

现在你可以使用 AV Fondation 来播放声音文件了，即使程序在后台，你的声音文件还是会被播放。

12.4.3. 讨论

在 iOS 中，即使程序在后台运行，他还是可以请求继续播放声音文件。AV Foundation 的 AVAudioPlayer 是个易于使用的播放器，我们将在本节中使用它。我们的任务是启动一个音频播放器并播放一首简单的歌曲，当歌曲在播放时，按下 Home 按钮将程序转入后台。如果我们在程序的.plist 文件中包含了 UIBackgroundModes 键，iOS 将继续播放音频播放器正在播放的音乐，即使程序在后台。当程序在后台时，程序应该只播放音乐以及日工音乐播放器运行时必要的数据。我们不勇敢运行其他任何的任务，比如显示新的屏幕等等。

下面是启动 AVAudioPlayer 的一个简单应用程序委托的,h 文件：

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface Playing_Audio_in_the_BackgroundAppDelegate
: UIResponder <UIApplicationDelegate, AVAudioPlayerDelegate>
@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, strong) AVAudioPlayer *audioPlayer;
@end
```

当我们的程序被打开时，我们会分配和初始化我们的音频播放器，读取一个名为 MySong.mp4 的文件的内容到 NSData 的实例中，并在我么的音频播放器的初始化过程中使用这个数据：

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
dispatch_queue_t dispatchQueue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch_async(dispatchQueue, ^(void) {
NSError *audioSessionError = nil;
AVAudioSession *audioSession = [AVAudioSession sharedInstance];
if ([audioSession setCategory:AVAudioSessionCategoryPlayback
error:&audioSessionError]){ NSLog(@"Successfully set the audio session.");
} else {
NSLog(@"Could not set the audio session"); }
NSBundle NSString
```

```
NSData
NSError
*mainBundle = [NSBundle mainBundle];
*filePath = [mainBundle pathForResource:@"MySong" ofType:@"mp3"];
*fileData = [NSData dataWithContentsOfFile:filePath]; *error = nil;
the audio player */
/* Start
self.audioPlayer = [[AVAudioPlayer alloc] initWithData:fileData
error:&error];
/* Did we get an instance of AVAudioPlayer? */ if (self.audioPlayer != nil){
/* Set the delegate and start playing */ self.audioPlayer.delegate = self;
if ([self.audioPlayer prepareToPlay] && [self.audioPlayer play]){
NSLog(@"Successfully started playing..."); } else {
NSLog(@"Failed to play."); }
} else {
/* Failed to instantiate AVAudioPlayer */ }
});
self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor]; [self.window makeKeyAndVisible];
return YES; }
```

请记住后台播放音频在 iPhone 模拟器上可能无效。你需要在真机上测试本小节。在模拟器上不同的是，一旦程序被送入后台音频将停止播放。

在这个例子中，在开始播放音频之前，我们使用 AV 音频会话来使其他程序（比如 iPod 程序）的音乐回放静音。关于音频会话的更多信息，请参考 9.5 小节。在后台时，你不仅限于播放当前的音频文件。如果当前在播放的音频文件（在后台）完成播放，你可以启动另一个 AVAudioPlayer 实例来播放一个全新的音频文件。iOS 将为这个处理请求作出调整，但在后台时，不能保证你的程序被允许分配出足够的内存，来容纳要播放的新的声音文件的数据。

另一个需要记住的重点是，当你的程序在后台运行一个音频文件时，UIApplication 的 backgroundTimeRemaining 属性的返回值将不会改变。也就是说，一个请求在后台播放音频文件的程序没有隐式或显示的向 iOS 请求额外时间。

12.4.4. 参考

12.5. 在后台处理位置变化

12.5.1. 问题

你在写一个程序，它的主要功能是使用 Core Location 来处理位置变化。你希望即使程序在后台时，也能获得 iOS 设备位置变化的通知。

12.5.2. 方案

在你的主程序.plist 文件（main application .plist file）中，向 UIBackgroundModes 键中添加 location 值，如下：

```
<dict> ...
...
... <key>UIBackgroundModes</key> <array>
<string>location</string> </array>
... ... ... </dict>
```

12.5.3. 讨论

当你的程序在前台运行时，你可以从一个 CLLocationManager 实例获得委托消息，在 iOS 检测到设备移动到新位置时告诉你这一变化。但是，如果你的程序被送到后台并且不再处于活动状态，这些位置委托消息一般是不会被发给你的程序的。实际上，他们会在你的程序回到前台后，批量的被发送。

如果在后台时，你仍然想能够获得用户设备位置改变的消息，你就必须在你的主程序.plist 文件（main application .plist file）中，向 UIBackgroundModes 键中添加 location 值，就像本小节的“方案”中指出的那样。这样一旦程序转到后台，你的程序将不断的获得设备位置变化的消息。让我们在一个仅有应用程序委托的简单程序中测试一下。

在这个应用中，我打算在 app 委托中保存一个布尔值，我将它取名为 executingInBackground。当程序退到后台，我将他设置为 YES，回到前台时，我会将它设置为 NO。当我们得到 Core Location 的位置更新消息，我会检查这个标志，如果此标志为 YES，我们就不做任何密集计算或者 UI 更新，因为，我们的程序在后台，作为一个负责的程序员，我们不应该在程序运行在后台时，做开销很大的处理。但是如果我们的应用在前台，我们就拥有设备所有的处理能力来做我们想做的一般处理。程序在前台或是后台时，我们也会尝试得到最佳的位置变化精度，我们将确定在位置更新时请求更低的精度，来减轻位置感应器的压力。让我们继续定义我们的 app 委托吧：

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
@interface Handling_Location_Changes_in_the_BackgroundAppDelegate
: UIResponder <UIApplicationDelegate, CLLocationManagerDelegate>
@property (nonatomic, strong) UIWindow *window;
@property (nonatomic, strong) CLLocationManager *myLocationManager;
@property (nonatomic, unsafe_unretained, getter=isExecutingInBackground) BOOL executingInBackground;
@end
```

然后，让我们自己同步我们的属性，并编写 executingInBackground 属性的读写方法，叫做 isExecutingInBackground：

```
#import "Handling_Location_Changes_in_the_BackgroundAppDelegate.h"
@implementation Handling_Location_Changes_in_the_BackgroundAppDelegate
@synthesize window = _window; @synthesize myLocationManager; @synthesize executingInBackground;
- (BOOL) isExecutingInBackground{ return executingInBackground;
}
...
```

现在，在我们程序启动时，继续创建和启动位置管理器：

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
self.myLocationManager = [[CLLocationManager alloc] init];
self.myLocationManager.desiredAccuracy = kCLLocationAccuracyBest;
self.myLocationManager.delegate = self; [self.myLocationManager startUpdatingLocation];
self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];
return YES; }
```

你可以看到，我们将位置管理器的要求精度设为了最高级别。但是，当我们退到后台时，希望降低这个精度来让 iOS 稍稍休息一下：

```
- (void)applicationDidEnterBackground:(UIApplication *)application{ self.executingInBackground = YES;
/* Reduce the accuracy to ease the strain on iOS while we are in the background */
self.myLocationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters; }
```

当我们的应用从后台激活时，我们可以将精度改到最高级别：

```
- (void)applicationWillEnterForeground:(UIApplication *)application{ self.executingInBackground = NO;
/* Now that our app is in the foreground again, let's increase the location detection accuracy */
self.myLocationManager.desiredAccuracy = kCLLocationAccuracyBest; }
```

而且，当程序运行在后台时，我们希望避免在从位置管理器得到新位置时进行密集处理，所以我们需要对位置管理器的 locationManager:didUpdateToLocation:fromLocation:委托方法做如下处理：

```
- (void)locationManager:(CLLocationManager *)manager didUpdateToLocation:(CLLocation *)newLocation
fromLocation:(CLLocation *)oldLocation{
if ([self isExecutingInBackground]){
/* We are in the background. Do not do any heavy processing */
} else {
/* We are in the foreground. Do any processing that you wish */
} }
```

简单的法则是，当我们在后台时，我们应该使用尽可能少量的内存和处理器能力来满足程序的需要。所以，通过在后台时降低位置管理器的精度，我们就减少了对 iOS 必须向我们程序发送新位置消息的处理。

根据你测试程序使用的不同的 iOS 模拟器版本，以及不同的网络设置和其他许多会影响这个过程的因素，后台处理可能是不可用的。请使用本小节提供的代码，在真机上测试你的程序。

12.5.4. 参考

12.6. 保存和加载多任务的 iOS 程序的状态

12.6.1. 问题

你希望在程序被送到后台时保存你 iOS 应用的状态，并在程序回到前台时恢复到和之前相同的状态。

12.6.2. 方案

使用发送到你的应用程序委托的 UIApplicationDelegate 协议消息，和 iOS 发送消息的组合，来保存你的多任务程序的状态。

12.6.3. 讨论

当一个空的 iOS 程序（仅有一个窗口没有其他任何代码的程序）在支持多任务的 iOS 设备上第一次运行时，下列的 UIApplicationDelegate 消息将被发送给你的应用程序委托，以下面的顺序：

```
1. application:didFinishLaunchingWithOptions:
2. applicationDidBecomeActive:
```

如果用户按下了 iOS 设备上的 Home 按钮，你的应用程序委托将收到按顺序下列消息：

```
1. applicationWillResignActive:
2. applicationDidEnterBackground:
```

一旦程序在后台，用户可以按下 Home 按钮两次，从后台程序列表中选择我们的程序。程序一旦再次回到前台，我们将以下面的顺序在应用程序委托中收到这些消息：

```
1. applicationWillEnterForeground:
2. applicationDidBecomeActive:
```

当我们的程序被送到后台或回到前台时，除了这些消息，我们还会从 iOS 收到各种通知消息。

为了保存和加载你的应用的状态，你需要细心考虑需要在转入后台时暂停并在回到前台时恢复的任务。让我给你个例子。如 12.7 小节所提到的，网络间接可以被系统本身轻易的恢复，所以我们大概不必再从网络下载文件的情况下做些特别的事情。但是，举个例子，如果你在写一个游戏，你最好在你的程序被送到后台时监听 iOS 发送的通知，然后相应的行动。在那种情况下，你可以简单的将游戏引擎置为暂停状态。如果有必要，你也可以将声音引擎暂停。

在程序被送往后台之后，你有 10 秒钟来保存任何为保存的数据，以及为任何时候用户将程序带回前台做好准备。如果需要的话，你可以选择要求更多的执行时间（关于这点更多的信息参见 12.2 小节）。

让我们用一个例子展示如何保存程序的状态。假设我们在为 iOS 编写一个游戏。当我们

的游戏被送到后台，我们想：

1.将游戏引擎置为暂停状态。

2.将用户的分数保存到磁盘。

3.保存当前关卡的数据到磁盘。这包括用户在关卡中的位置，关卡的物理参数，照相机位置，等等。

当用户重新打开程序，将程序带回前台，我们想：

1.从磁盘加载用户的分数。

2.从磁盘加载上次用户在玩的关卡。

3.恢复游戏引擎。

现在假设我们的应用程序委托就是游戏引擎。然我们在它的头文件中定义一些方法：

```
#import <UIKit/UIKit.h>
@interface Saving_and_Loading_the_State_of_Multitasking_iOS_AppsAppDelegate : UIResponder <UIApplicationDelegate>
@property (strong, nonatomic) UIWindow *window;
/* Saving the state of our app */ - (void) saveUserScore;
- (void) saveLevelToDisk;
- (void) pauseGameEngine;
/* Loading the state of our app */
- (void) loadUserScore;
- (void) loadLevelFromDisk; - (void) resumeGameEngine;
@end
```

在我们的应用程序委托中我们将对这些方法进行站位实现（place stub implementations）。

```
#import "Saving_and_Loading_the_State_of_Multitasking_iOS_AppsAppDelegate.h"
@implementation Saving_and_Loading_the_State_of_Multitasking_iOS_AppsAppDelegate
@synthesize window = _window;
- (void) saveUserScore{
/* Save the user score here */
}
- (void) saveLevelToDisk{
/* Save the current level and the user's location on map to disk */
}
- (void) pauseGameEngine{
/* Pause the game engine here */
}
- (void) loadUserScore{
/* Load the user's location back to memory */ }
- (void) loadLevelFromDisk{
/* Load the user's previous location on the map */ }
- (void) resumeGameEngine{
/* Resume the game engine here */ }
...
```

现在我们需要确定我们的程序能够处理中断，比如在 iPhone 上接到来电。在那种情况下，我们的应用不会被送到后台，而是会变为非激活状态。比如，当用户结束通话，iOS 会将程序重新设置为激活状态。所以当我们的程序变为非激活时，我们要确保正在暂停游戏引擎，当程序重新激活时，我们可以恢复游戏引擎。我们不需要在程序变为非激活时向磁盘保存任何东西（至少在本例中），因为 iOS 将在程序重新激活时将它置为之前的状态：

```
- (void)applicationWillResignActive:(UIApplication *)application{
[self pauseGameEngine]; }
- (void)applicationDidBecomeActive:(UIApplication *)application{ [self resumeGameEngine];
}
```

现在，当我们的应用被送到后台，我们将保存游戏的状态，在程序回到前台时，我们将重新加载程序状态：

```
- (void)applicationDidEnterBackground:(UIApplication *)application{
[self saveUserScore]; [self saveLevelToDisk]; [self pauseGameEngine];
}

- (void)applicationWillEnterForeground:(UIApplication *)application{ [self loadUserScore];
[self loadLevelFromDisk];
[self resumeGameEngine];
}
```

不是所有程序都是游戏。但是，在 iOS 的多任务环境中，你可以使用这项技术来保存和加载程序的状态。

12.6.4. 参考

12.2 小节

12.7. 在后台处理网络连接

12.7.1. 问题

你在使用 NSURLConnection 的实例从一个 web 服务器收发数据，并在考虑在 iOS 的多任务环境中，如何让你的程序不会出现连接失败。

12.7.2. 方案

确保你在你提交给连接对象的块对象（block objects ）中支持连接失败。

12.7.3. 讨论

对于使用 NSURLConnection 但没有向 iOS 借用额外时间的程序，在他们被送到后台时，连接处理真的很简单。让我们通过一个例子来看看，在程序被送到后台和被再次带回前台时，一个异步连接如何行动。为此，让我们让我们发送一个异步连接请求来获取某个 URL 的内容，假设，是 Apple 的首页：

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url]; NSOperationQueue *queue = [[NSOperationQueue alloc]
init];
```

```
[NSURLConnection
sendAsynchronousRequest:urlRequest
queue:queue
completionHandler:^(NSURLResponse *response, NSData *data, NSError *error) {
if ([data length] > 0 &&
error != nil){
/* Date did come back */
}
else if ([data length] == 0 &&
error != nil){ /* No data came back */
}
else if (error != nil){
/* Error happened. Make sure you handle this properly */ }
}];
self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];
return YES; }
```

我建议你用因特网上另一个指向更大文件的 URL 来替换 Apple 的首页。因为如果你的程序正在下载一个大文件，你就有更多的时间来操作程序，将其送往后台和带回前台。而当你的因特网连接相当快，并且你只是下载 Apple 的首页，那么连接将在一两秒内收完数据。

在前台时，我们的程序将持续下载这个文件。下载时，用户可以按下 Home 按钮，将程序送到后台。你所见的将会相当神奇！iOS 会自动将下载进程置为暂停状态。当用户将程序重新带回前台，你不需要写一行代码来处理多任务，下载会自动恢复。

现在让我们看看同步连接发生了什么。在程序启动的同时，我们会在主线程中下载一个很大的文件（一个很糟的做法，别在产品级别的程序中这么做）：

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
/* Replace this URL with the URL of a file that is rather big in size */ NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url]; NSError *error = nil;
NSData *connectionData = [NSURLConnection sendSynchronousRequest:urlRequest returningResponse:nil
if ([connectionData length] > 0 && error == nil){
}
else if ([connectionData length] == 0 &&
error == nil){
}
else if (error != nil){
}
self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor]; [self.window makeKeyAndVisible];
return YES;
}
```

如果你的程序被送到后台，你会注意到程序的 GUI 被送到了后台，但是程序的核心没有被送到后台，而适当的委托方法 --applicationWillResign Active: 和 applicationDidEnterBackground: --- 将永远不会被调用。我在 iPhone 上做过了这个测试。

这种做法的问题是，我们用同步下载文件的方式，消耗了主线程的时间片（time slice）。我们可以通过在主线程中异步下载这些文件，或者在单独的线程中执行同步下载，来解决这个问题。

根据之前的示例代码，如果我们在一个全局并发队列中下载同一个大文件，在程序被送到后台时连接会被暂停，并在程序再次回到前台时恢复：

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
dispatch_queue_t dispatchQueue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch_async(dispatchQueue, ^(void) {
/* Replace this URL with the URL of a file that is rather big in size */ NSString *urlAsString = @"http://www.apple.com";
NSURL *url = [NSURL URLWithString:urlAsString];
NSURLRequest *urlRequest = [NSURLRequest requestWithURL:url];
NSError *error = nil;
NSData *connectionData = [NSURLConnection sendSynchronousRequest:urlRequest returningResponse:nil
if ([connectionData length] > 0 && error == nil){
}
else if ([connectionData length] == 0 &&
error == nil){
}
else if (error != nil){
} });
self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];
return YES; }
```

12.7.4. 参考

12.2 小节

12.8. 处理发送给一个被唤醒的程序（Waking App）的通知

12.8.1. 问题

当你的程序被带回到前台，你希望能够得到关于重要系统变化的通知，比如用户区域变化。

12.8.2. 方案

只要监听 iOS 发送给唤醒中程序的许多系统通知其中之一。下面列出了其中的一些通知：

NSCurrentLocaleDidChangeNotification

这个通知在用户改变区域设置时发送，比如，用户在设备的设置页，将 iOS 设备的语言从英文改为西班牙文。

NSUserDefaultsDidChangeNotification
这个通知在用户在 iOS 设备的设置页(如果提供给用户任何设置项)改变了程序设置时触发。

UIDeviceBatteryStateDidChangeNotification
这个通知在 iOS 设备上的电池状态变化时都会被发送。比如，当程序在前台时设备被接入到一台电脑，然后在后台被拔出时，程序会接收到这一通知（如果程序注册了此通知）。电池状态可以使用 UIDevice 的 batteryState 属性获得。

UIDeviceProximityStateDidChangeNotification
当接近感应器改变时会发送此通知。最后的状态可以通过 UIDevice 的 proximatyState 属性获得。

12.8.3. 讨论

当你的程序在后台时，很多事情都可能发生！比如，用户可能会突然在设备上的设置页，将 iOS 的区域设置从英文改为西班牙文。程序可以为这些通知注册自己。通知可以被合并然后一起发给唤醒中的程序。让我解释一下什么叫做“合并”吧。假设你的程序在前台，而你已经注册了 UIDeviceOrientationDidChangeNotification 通知。现在，你的用户按下了 Home 按钮，你的程序被送到后台。然后用户旋转设备从竖屏，到横屏（右），然后到竖屏，然后到横屏（左）。当用户将你的程序带回到前台后，你只会收到一个 UIDeviceOrientationDidChangeNotification 类型的通知，这叫做合并。所有其他在程序被打开之前发生的横竖屏变化都不重要（因为你的程序不在屏幕上），系统不会将他们发给你的程序。但是，系统将为系统的每个方面向你至少发送一个通知，比如方向变化，然后你可以检测到设备最近一次方向变化。

这里有一个简单的视图控制器的实现，得益于这项技术来决定横竖屏的变化：

```
#import "Handling_Notifications_Delivered_to_a_Waking_AppViewController.h" @implementation
Handling_Notifications_Delivered_to_a_Waking_AppViewController
- (void) orientationChanged:(NSNotification *)paramNotification{
NSLog(@"Orientation Changed"); }
- (void)viewDidLoad{ [super viewDidLoad];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(orientationChanged:)
name:UIDeviceOrientationDidChangeNotification object:nil];
}

- (void)viewDidUnload{ [super viewDidUnload];
[[NSNotificationCenter defaultCenter] removeObserver:self]; }
- (BOOL)shouldAutorotateToInterfaceOrientation :(UIInterfaceOrientation)interfaceOrientation{
return YES; }
@end
```

现在在设备上运行该程序。当视图控制器在屏幕上显示时，按下 Home 按钮将程序送入后台。现在尝试多次的改变设备的方向，然后再次打开程序。观察结果，你会看到在你程序打开的时候，最多有一个通知会被送到 orientationChanged:方法。当然，如果你的视图结构支持横竖屏切换，你可能得到第二次调用。

12.8.4. 参考

12.9. 响应应用设置的变化

12.9.1. 问题

你的程序向用户暴露了一个设置目录。你想在用户对你程序的设置作出改变后（当程序在后台时），一旦程序回到前台立即得到通知。

12.9.2. 方案

注册一个 NSUserDefaultsDidChangeNotification 通知。

12.9.3. 讨论

为 iOS 编写的程序可以暴露一个关于设置的目录文件(bundle file)。用户可以通过设备上的 Setting 程序来访问这些设置。为了更好的理解这点，让我们创建一个设置目录：

1。在 Xcode 中，选择 File-->New File
2.确定 iOS 类别在左侧被选择。
3.选择 Resuorces 子类别。
4.选择 Setting Bundle 作为文件类型并点击 Next。
5.设置文件名为 Settings.bundle。
6.点击 Save。

现在在 Xcode 中有一个名为 Settings.bundle 的文件了。让文件保持原样，不要修改它。将本节“方案”中的代码放入你的根视图控制器中，运行程序。按下设备上的 Home 按钮转到设备上的 Setting 程序，如图 12-6 所示（我创建的例子程序名为"Responding to Changes in App Settings"）。

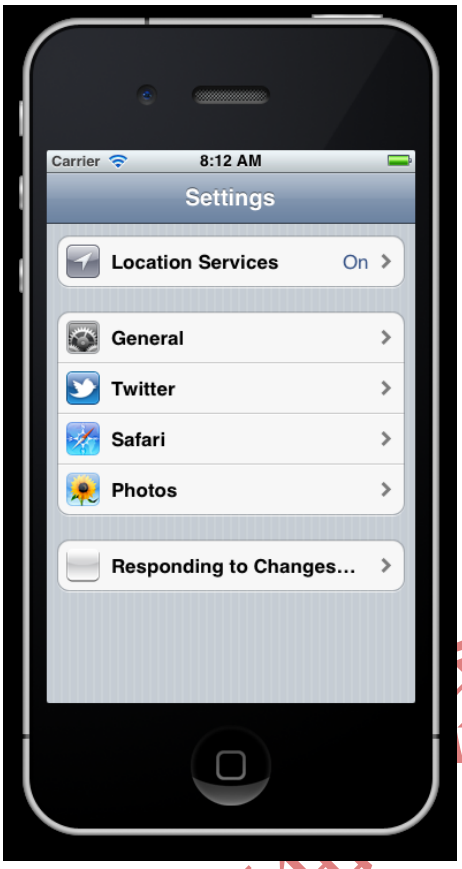

图 12-6. 在 iOS 模拟器的 Setting 应用中显示的 Setting.bundle

点击你的程序名字来查看你程序暴露给用户的设置项，如图 12-7 所示。

图 12-7. 默认的 Settings.bundle 的内容

让我们继续在应用程序委托中，开始监听 NSUserDefaultsDidChangeNotification 通知，当程序终止时，显然，我们要将应用程序委托从通知链中移除：

```
#import "Responding_to_Changes_in_App_SettingsAppDelegate.h" @implementation
Responding_to_Changes_in_App_SettingsAppDelegate @synthesize window = _window;
- (void) settingsChanged:(NSNotification *)paramNotification{
NSLog(@"Settings changed");
NSLog(@"Notification Object = %@", [paramNotification object]);
}
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions{
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(settingsChanged:) name:NSUserDefaultsDidChangeNotification object:nil];
self.window = [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible];
return YES; }
- (void)applicationWillTerminate:(UIApplication *)application{
[[NSNotificationCenter defaultCenter] removeObserver:self]; }
@end
```

现在，当程序在后台时，尝试修改这些设置，在你完成修改后，将程序带回前台，你会看到 NSUserDefaultsDidChangeNotification 消息将被发送到你的程序。和通知一并发过来的对象将会是 NSUserDefaults 类型的对象，并且将包含你的程序设置中的 userdefaults。

12.9.4. 参考

12.10. 关闭多任务

12.10.1. 问题

你不希望你的程序以多任务方式运行

12.10.2. 方案

在你程序的主 plist 文件中添加 UIApplicationExitsOnSuspend 键，并设置其值为 true

```
<dict> ...
... ...
<key>UIApplicationExitsOnSuspend</key>
<true/>
... ...
... </dict>
```

12.10.3. 讨论

在某些情况下，你大概需要你的 iOS 程序不使用多任务（虽然，我强烈推荐你让程序支持多任务）。在那样的情况下，你可以向你的程序的主 plist 文件中添加 UIApplicationExitsOnSuspend 键。使用最新版本支持多任务的 OS 的设备明白这个值的含义，当程序主 plist 文件中的这个值为 true 时，OS 会终止程序。在早期的 iOS 版本中不支持多任务，这个值对操作系统就没什么意义了，会被忽略掉。

当这样的程序在最新的 iOS 上运行时，下列应用程序委托消息会被发送到你的程序：

```
1. application:didFinishLaunchingWithOptions:
2. applicationDidBecomeActive:
```

如果用户按下了设备上的 Home 按钮，下列消息将被发送给你的应用程序委托：

```
1. applicationDidEnterBackground:
 2. applicationWillTerminate:
```

12.10.4. 参考