从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter26

开发环境：
Xcode 5 +iOS 7

欢迎继续我们的学习。

虽然这款应用已经很完整了，不过最后我们还送上一个额外的特性教程-local notification(本地消息通知）。

通过使用本地消息通知，可以让应用安排一个时间点通知用户，即便应用没有应用也不受影响。我们可以在ChecklistItem对象中添加一个"截止时间"字段，然后在deadline截止时间快到的时候向用户发出通知来提醒。

如果你对这个功能比较感兴趣，可以继续看下去。
如果不感兴趣，可以就此结束系列2教程的学习，开始新系列教程的学习，或是直接投入到项目实战之中。

好了，这部分的内容将会包括：

1.尝试使用local notification，然后看是否可以顺利工作
2.允许用户为待办事项设置一个截止日期
3.创建一个日期选择控制器
4.为待办事项计划安排本地消息通知，并且当用户更改了截止日期的时候更新安排

首先让我们来尝试第一步，也就是安排一个本地消息推送，看看会发生些神马。

需要注意的是，local notification（本地消息通知）和push notification（消息推送）是不同的概念。消息推送可以让你的应用从外部事件中接收信息，比如你最喜欢的阿根廷足球队在巴西世界杯上夺冠。而本地消息通知则类似于闹钟设置：我们可以设置一个特定的时间，然后此时它会提醒用户该干嘛干嘛。

打开Xcode,切换到ChecklistsAppDelegate.m，然后在application:didFinishLaunchingWithOptions:方法中添加以下代码：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{

    //data model setting

    _dataModel = [[DataModel alloc]init];

    UINavigationController *navigationController = (UINavigationController *)self.window.rootViewController;
```

```
    AllListsViewController *controller = navigationController.viewControllers[0];

    controller.dataModel = _dataModel;

    //local notification

    NSDate *date = [NSDate dateWithTimeIntervalSinceNow:10];

    UILocalNotification *localNotification = [[UILocalNotification alloc]init];

    localNotification.fireDate = date;
    localNotification.timeZone = [NSTimeZone defaultTimeZone];
    localNotification.alertBody = @"2014年到了，马上有钱！";
    localNotification.soundName = UILocalNotificationDefaultSoundName;


    [[UIApplication sharedApplication]scheduleLocalNotification:localNotification];

    return YES;
}
```

还记得吗？当应用启动的时候会立即调用didFinishLaunchingWithOptions方法。这里我们创建了一个新的本地消息通知对象，然后告诉它在应用启动10秒钟后开火，哦不，发出提醒。

这里我们使用NSDate对象来指定 了一个特定的日期和时间。我们使用dateWithTimeIntervalSinceNow这个便捷构造方法来创建了一个NSDate对象，它的准确时间是应用启动后10秒。

当我们创建UILocalNotification对象时，将其fireDate属性设置为刚才的NSDate对象。此外我们还设置了所在时区，这样系统会根据时区变化自动调整开火时间（如果用户是经常出差的空中飞人，这会很有帮助的）。

我们有不同的方式来显示本地消息通知。这里我们设置了一个文本，当通知发出的时候会显示一个提示消息。这里还设置了声音。最后，我们让UIApplication对象（也就是这款应用）安排了本地消息通知）。

理论知识充电-关于UIApplication

之前我们从未接触过这个对象，但实际上任何一个iOS应用都提供了这样的一个对象，它可以处理和整个应用相关的功能。我们需要为UIApplication对象提供一个代理对象，以处理类似applicationDidEnterBackground这样的消息。对于我们这个应用来说，UIApplication的代理对象就是ChecklistsAppDelegate。默认情况下，Xcode会为每个应用提供一个app delegate。
虽然在开发的过程中通常不会过多接触到UIApplication，但在涉及到类似本地消息通知等特殊功能的时候，UIApplication对象还是很有用的。

接下来在ChecklistsAppDelegate.m中添加一个方法如下：

```
-(void)application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification *)notification{

    NSLog(@"didReceivedLocalNotification %@",notification);
}
```

当本地消息通知发出后，如果应用在运行状态，或是在后台悬停状态，就会触发该方法。当然这里我们什么也没做，只是在debug调试面板区输出一行文本。
对于某些应用来说，需要对本地消息通知做出一些响应，比如向用户显示一行消息，或是刷新界面等等。

好了，此时编译运行应用。等应用启动后立即在Simulator中选择hardware-Home（如果在设备商直接按Home键）。等待10秒左右。

10秒钟过后会在Notification Center看到类似下面的显示：

触碰这行消息，就会自动返回应用。

此时在Xcode的调试信息区可以看到类似下面的消息：

```
2013-12-31 10:49:10.038 Checklists[1554:70b] didReceivedLocalNotification
<UIConcreteLocalNotification: 0x1091ae220>{fire date = Tuesday, December 31, 2013 at
10:48:05 AM China Standard Time, time zone = Asia/Chongqing (GMT+8) offset 28800, repeat
interval = 0, repeat count = UILocalNotificationInfiniteRepeatCount, next fire date =
(null), user info = (null)}
```

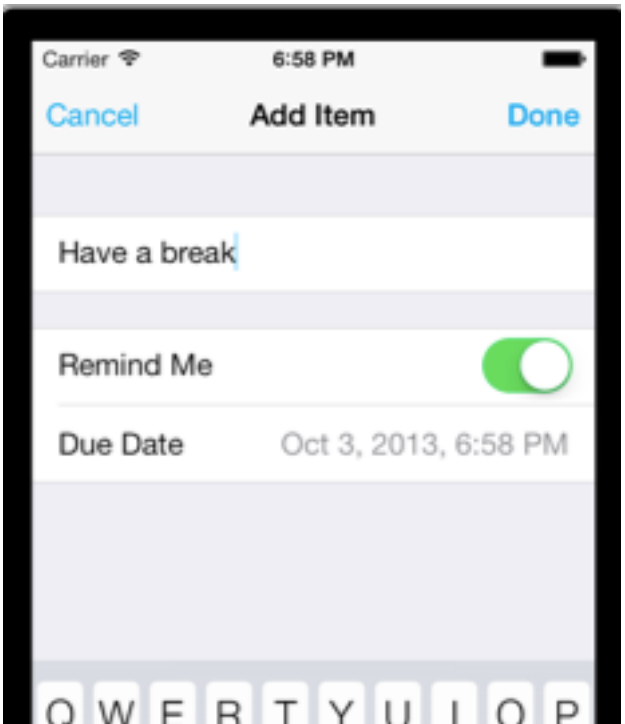
为什么这里要求你触碰Home键呢？因为iOS只有在应用并非处于活跃状态时才会显示通知消息的提示。

好了，为了验证这一点，可以关闭应用再次运行。
现在不要触碰Home键，在那里数羊慢慢等待。

10秒钟后，我们会在Xcode的debug调试信息区看到didReceiveLocalNotification的消息，但不会看到提示信息。（不过在Notification Center中可以看到通知消息）。如果应用在前台运行，就会用自己的方式来处理任何被触发的消息。

好了，现在我们已经看到本地消息通知的作用了。
不过暂时还是将ChecklistsAppDelegate.m恢复原貌，因为这里并不希望在用户启动应用时每次都安排一个消息通知。
让我们将didFinishLaunchingWithOptions方法代码恢复如此。
至于didReceiveLocalNotification方法可以保留在那里，至少对于调试还有帮助。
记得恢复之后编译运行下，小心别删错了东西。
至此，第一步已经顺利完成，搞定收工。

接下来让我们想一想在应用中应如何处理本地消息通知。每个ChecklistItem都会获得一个due date(截止日期）字段，也就是一个NSDate对象，然后还有一个BOOL属性变量，用来保存用户是否希望该项目被提醒。
大家每天都要做N多事，显然我们不希望每件事都被提醒一下，对于这些不希望被提醒的事项，最好就不要安排本地消息通知了。这种BOOL变量通常被称为flag，让我们将其命名为shouldRemind>
当我们添加了这两个字段后，Add/Edit Item界面的显示将类似下面：



其中Due Date（截止日期）字段需要提供日期选择器控件。在iOS中提供了一个很酷的日期选择器视图，后续我们将会把它添加到表视图中。

好吧，现在来想想应该如何以及何时来安排本地消息通知。

1.当用户添加一个新的ChecklistItem对象时，如果它设置了一个shouldRemind 标志，那么就应该安排一个新的消息通知。

2.当用户更改了现有ChecklistItem的截止日期时，旧的消息通知会被取消（如果有的话），然后创建一个新的消息通知（如果该代办事项有一个shouldRemind标志）。

3.当用户关闭了现有ChecklistItem的shouldRemind标志时，就会取消当前的消息通知。反之，如果用户开启了shouldRemind标志，就需要创建一个新的消息通知。

4.当用户删除了一个ChecklistItem时，如果它有一个消息通知，就应该被取消。

5.当用户删除了整个Checklist时，那么该Checklist中所有ChecklistItem的消息通知都应该被取消。

这样看来这事还不是一般的麻烦，不过至少也理清了头绪，接下来也好知道具体怎么办。

看来我们需要不仅需要创建新的消息通知，还需要知道如何取消这些通知。我们还需要检查一下，如果某些代办事项的due date截止日期发生在遥远的过去（穿越？），那么就不应创建消息通知。虽然iOS很聪明，通常会忽略这一类型的通知，但对于应用开发者来说还是要谨慎为妙。

UIApplication有一个cancelLocalNotification方法，可以取消一个之前所设置的消息。该方法需要获得某个UILocalNotification对象做参数。为了取消某个消息通知，我们必须将ChecklistItem对象和UILocalNotification关联在一起。

此时UILocalNotification对象会保存到ChecklistItem对象中，这样看起来比较合理。但如果应用切换到后台悬停状态肿么办？此时我们可以将ChecklistItem对象保存到Checklists.plist文件中，但UILocalNotification对象怎么办？因为UILocalNotification也遵循NSCoding协议，看起来似乎我们应该让它和ChecklistItem一起保存到plist文件中。
不过这样做实际上是自讨麻烦。
UILocalNotification对象是操作系统所拥有的，并不属于我们的应用。当应用再次启动的时候，很有可能iOS会使用不同的对象来代表同一个消息通知。我们不能从plist文件中解冻这些对象，然后指望iOS系统可以认出它们。

因此，不要考虑直接保存UILocalNotification对象。

这里我们还有一种更好的替代方案，就是给UILocalNotification对象提供一个到所关联的ChecklistItem对象的引用。每个本地消息通知对象都有一个名为userInfo的NSDictionary属性，我们可以用它来保存自己的定制信息。

当然，直接用词典来保存ChecklistItem对象显然是不明智的，原因很明显：
当应用关闭后重新打开时，会重新创建新的ChecklistItem对象。即便这些对象从长相到行为方式和之前的ChecklistItems完全相同，它们也可能被保存到内存的其它地方，此时UILocalNotifications中的引用就会失效。

想象一下科幻小说《副本》中的场景，你在地球上上传保存了思维，然后几百万年后在遥远的河外星系上的某个星球上被"复活"。虽然从哲学的角度来看你和之前一样，但此前那段"史前生活"中所创建的社会联系却完全失效了。

因此，我们不会采用直接的引用，而是改用一个数字化的标识符。我们将赋予每个ChecklistItem对象一个独特的数字化ID。在创建数据模型时为对象分别数字化ID是常见的做法，它就类似给一个关系数据库中的记录提供一条数字化主键。

你为什么是你，而不是别人？我们需要根据你的DNA排列，脑神经组合等信息综合创建一个数字化ID。即便你当前的"化身"不复存在，但可以根据这个数字化ID重新"复活"一个全新的你，并把你之前的记忆和人格重新赋予你。

我们将把这个ID保存到Checklists.plist文件中，同时还会在UILocalNotification的userInfo中保存。接下来就可以根据ChecklistItem对象轻松找到对应的本地消息，或是根据本地消息找到对应的ChecklistItem对象。

即便应用被强关，所有的初始对象被销毁很久之后，我们仍然得以完整恢复这些信息。

好了，之所以这里用上这么多废话来讲这个事情，是因为-

永远先思考再编程！

哪怕你完全不懂如何写代码，起码也要知道该怎么去做的思路，然后就可以交给所谓的"码农"和"码奴"来具体搞定了。哪怕你看完所有的教程仍然不懂写代码，但一定要学会如何解决问题的思路，这才是学习编程的真正意义所在！

如果你自己都想不明白，就别指望计算机可以帮你搞定，毕竟它们的智慧水平也就是一个超级计算器的档次。
人类的价值应体现在自己的创造力、想象力和解决新问题的能力上，而不是试图和机器比赛记忆力和重复计算的能力。
机器擅长重复计算这种体力活，而人类则擅长创造性的思维，千万别搞反了，否则等机器人遍地的那一天就是你彻底失业的时候！
永远记住，不要当任何外物的奴隶，不管是机器还是人！
而要做到这一点，你必须学会用创造性的思维去解决问题，而不是像机器和奴隶一样被动接受安排

在Xcode中切换到ChecklistItem.h，并添加以下几个属性变量的声明：

```
@property(nonatomic,copy)NSDate *dueDate;
@property(nonatomic,assign) BOOL shouldRemind;
@property(nonatomic,assign) NSInteger itemId;
```

注意这里我们的拼法是itemId,而不是itemID.当然，这样写有个人的偏好在里面。不过有一点需要特别注意，千万不要直接用id，因为id 是Objective-C中的保留关键字，它和标识符没有任何关系。比如当我们创建自己的代理协议时就会用到id这个关键字。

好了，接下来需要扩展initWithCoder和encodeWithCoder方法，从而可以让这些新属性和ChecklistItem对象一起被保存。

切换到ChecklistItem.m，更改以下两个方法的代码如下：

```
- (id)initWithCoder:(NSCoder *)aDecoder
{
  if ((self = [super init])) {
    self.text = [aDecoder decodeObjectForKey:@"Text"];
    self.checked = [aDecoder decodeBoolForKey:@"Checked"];
     self.dueDate = [aDecoder decodeObjectForKey:@"DueDate"];
     self.shouldRemind = [aDecoder decodeBoolForKey:@"ShouldRemind"];
     self.itemId = [aDecoder decodeIntegerForKey:@"ItemID"];
  }
  return self;
}

- (void)encodeWithCoder:(NSCoder *)aCoder
{
  [aCoder encodeObject:self.text forKey:@"Text"];
  [aCoder encodeBool:self.checked forKey:@"Checked"];
   [aCoder encodeObject:self.dueDate forKey:@"DueDate"];
   [aCoder encodeBool:self.shouldRemind forKey:@"ShouldRemind"];
   [aCoder encodeInteger:self.itemId forKey:@"ItemID"];
}
```

通过上面的工作，我们就可以实现对现有对象的保存和加载。
不过我们还需要为新创建的对象来分配ID。

在ChecklistItem.m中添加一个常规的init方法如下：

```
-(id)init{

    if((self =[super init])){

        self.itemId = [DataModel nextChecklistItemId];

    }
    return self;
}
```

每当我们创建一个新ChecklistItem对象时，我们都要求DataModel对象提供一个新的item ID。

显然这里还需要在文件顶部添加一行代码：

```
#import “DataModel.h"
```

接着我们需要在DataModel中添加一个新的nextChecklistItemId方法。
切换到DataModel.h，先添加一个方法声明：
+(NSInteger)nextChecklistItemId;
然后切换到DataModel.m，添加该方法的实现代码：

```
+(NSInteger)nextChecklistItemId{

    NSUserDefaults *userDefaults =[NSUserDefaults standardUserDefaults];

    NSInteger itemId = [userDefaults integerForKey:@"ChecklistItemId"];

    [userDefaults setInteger:itemId +1 forKey:@"ChecklistItemId"];
    [userDefaults synchronize];
    return itemId;

}
```

这里我们又碰到老朋友NSUserDefaults了。在上面的方法中，我们首先从NSUserDefaults中获取了当前的ChwecklistItemId值，然后在该值的基础上加1，然后重新保存到NSUserDefaults中。最后将该值返回到调用该方法的对象。

同时该方法还用到了[userDefaults synchronize]来强制要求NSUserDefaults将这些改立即写入磁盘。这样当应用被强关时信息就不可能丢失了。
这一点至关重要，因为我们不需要有两个甚至更多的ChecklistItem拥有同样的ID.

接下来在registerDefaults方法中为ChecklistItemId添加一个默认的初始值：

```
-(void)registerDefaults{

    NSDictionary *dictionary =
@{@"ChecklistIndex" :@-1,@"FirstTime":@YES,@"ChecklistItemId":@0};

    [[NSUserDefaults standardUserDefaults]registerDefaults:dictionary];
}
```

当首次调用nextChecklistItemId方法的时候，会返回ID 0.下一次则是1，再下一次则是2，以此类推。

理论充电-类方法 vs 实例方法

刚才我们用到了一种新的形式来编写方法，

```
+(int)nextChecklistItemId
```

而不是
```
-(int)nextChecklistItemId
```

如果你注意到了这一点，恭喜你！看起来你仍然属于人类~
使用+而不是-，意味着我们可以在没有到DataModel对象引用的前提下仍然可以调用该方法。

记住，我们在ChecklistItem.m的初始化init方法中用的是：
```
self.itemId = [DataModel nextChecklistItemId];
```

而不是：
self.itemId = [self.dataModel nextChecklistItemId];

这是因为ChecklistItem对象没有到DataModel对象的引用，它们没有一个self.dataModel属性。虽然我们可以手动添加一个属性从而提供一个到DataModel对象的引用。不过这里用class method（类方法）显然要更方便一些。

类方法在iOS中用+（加号）开头，此类方法的作用对象是整个类。到目前为止我们接触的方法绝大多数都是instance method(实例方法）。实例方法以-（减号）开头，作用对象是该类的一个实例对象。

考虑到教程的循序渐进，这里暂时不对类方法和实例方法的区别做过多解释，在下一系列的教程中会详细解释。
暂时你只要记住+开头的方法可以允许你调用某个对象的方法，即便没有到该对象的引用也不会影响。

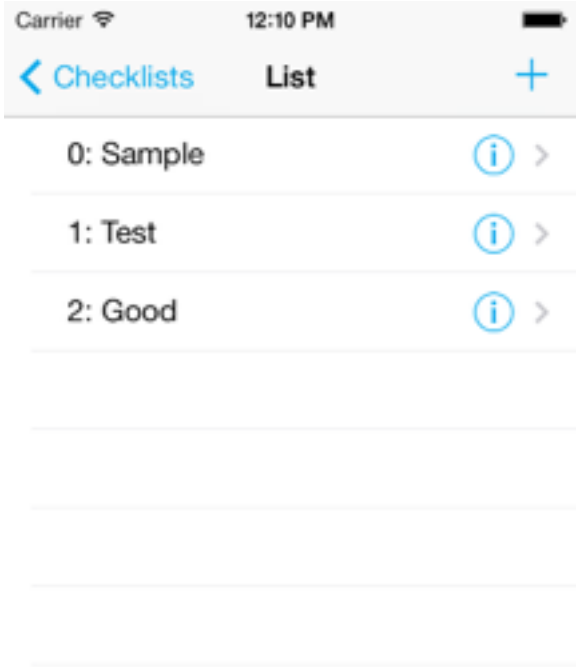
为了快速测试为ChecklistItem分配的ID是否起作用，我们将把它放在ChecklistItem cell的文本标签中。
当然，这只是临时的举措，因为对用户来说，根本不会在乎这些对象的内部编号是神马。

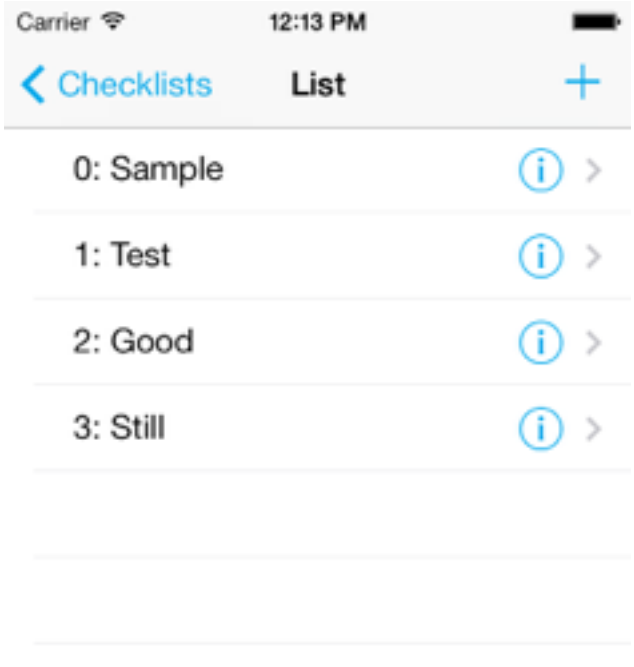
在Xcode中切换到ChecklistViewController.m，更改configureTextForCell方法如下：

```
- (void)configureTextForCell:(UITableViewCell *)cell withChecklistItem:(ChecklistItem *)item
{
  UILabel *label = (UILabel *)[cell viewWithTag:1000];
//  label.text = item.text;
    label.text = [NSString stringWithFormat:@"%ld: %@",(long)item.itemId,item.text];

}
```

这里将之前设置标签文本的语句注释了，因为后续还会重新启用的。
新的代码行中使用stringWithFormat方法将待办事项的itemId数学添加到标签文本中。
在编译运行应用之前，记得先从Simulator中将其删除。
因为Checklists.plist文件的组织格式再次发生了变化，这一点在测试时一定要多加注意。
编译运行应用，然后试着添加一些代办事项。现在每个代办事项都有了一个独特的标识符。从Simulator的菜单中选择Hardware-Home，然后从Xcode中停止应用。
再次编译运行应用，会看到新代办事项的编号会在之前的编号基础上继续。





好了，现在ID已经搞定了。
接下来我们将要在Add/Edit Item界面中添加due date(截止日期）和should remind(是否提醒）。
不过显然这部分的内容将会在明天的教程中出现，否则就会超出45分钟的平均学习时间了。

马年，祝大家新年快乐！