从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter28（终结章）

开发环境：
Xcode 5 +iOS 7

终于，在2014年新年的第2天，迎来了这一系列教程的终章。

当然，目前来说只能算是该系列教程翻译改写的1.0版本，后续还需要：
1.根据大家的反馈对内容改写
2.整理大家的问题反馈
3.开启新的篇章

首先还是让我们做一个了结吧·

别忘了最后一部分的工作-计划安排本地消息通知。

面向对象编程的一个原则是，让对象尽可能自力更生，独立自主。你可以把这些对象想象成生活在虚拟世界中的生命，他们有自己的喜怒哀乐，自己对于色彩的偏好，自己的24重人格。
至少对于ChecklistItem对象来说，他们应该具备自己安排计划消息通知的能力。

在Xcode中切换到ChecklistItem.h，添加一个方法声明：

```
-(void)scheduleNotification;
```

然后在ChecklistItem.m中添加该方法的实现代码：

```
-(void)scheduleNotification{
   if(self.shouldRemind &&
      [self.dueDate compare:[NSDate date]] != NSOrderedAscending){

       NSLog(@"需要安排一个消息通知");
   }

}
```

在上面的方法中，我们会将待办事项的截止日期和当前日期进行对比，如果截止日期已经过去了，就什么也不需要做。注意这里的&&操作符，也就是传说中的逻辑与。只有当Remind Me开关处于on的状态，而且截止日期在未来的时候才需要执行下面的操作。
当然，如果你是穿越过来的，或许就可以彻底免了这些事。

当用户在Add/Edit Item界面中触碰了done按钮的时候就可以调用该方法。

切换到ItemDetailViewController.m，更改done动作方法的代码如下：

```
- (IBAction)done
{
  if (self.itemToEdit == nil) {
    ChecklistItem *item = [[ChecklistItem alloc] init];
    item.text = self.textField.text;
    item.checked = NO;
    item.shouldRemind = self.switchControl.on;
    item.dueDate = _dueDate;

    [item scheduleNotification];

    [self.delegate itemDetailViewController:self didFinishAddingItem:item];

  } else {
    self.itemToEdit.text = self.textField.text;
    self.itemToEdit.shouldRemind = self.switchControl.on;
    self.itemToEdit.dueDate = _dueDate;

    [self.itemToEdit scheduleNotification];
    [self.delegate itemDetailViewController:self didFinishEditingItem:self.itemToEdit];
  }
}
```

在上面的方法中，我们通过黄色高亮的代码行调用了scheduleNotification方法。

现在编译运行应用，试试看。添加一个新项目，将Remind Me开关开启，但是不更改截止日期，触碰done。此时debug调试区不会有信息出现。

再添加一个代办事项，将Remind Me开关开启，然后选择未来的某个截止日期。当你触碰done按钮的时候，就会有一个NSLog信息在debug调试区提示。

好了，现在我们已经知道在何处调用这个方法了，接下来就要真正创建一个新的UILocalNotification对象。

首先考虑添加一个新代办事项的时候。

在ChecklistItem.m中更改scheduleNotification方法：

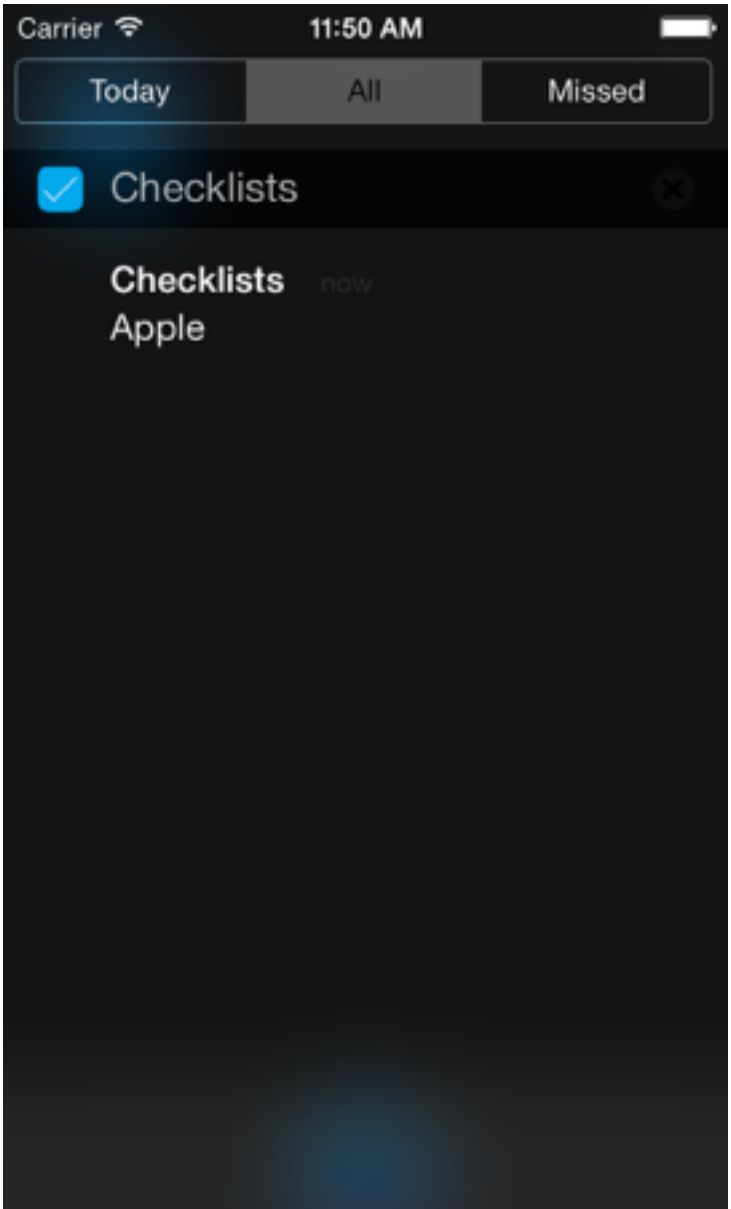
```
-(void)scheduleNotification{
    if(self.shouldRemind &&
       [self.dueDate compare:[NSDate date]] != NSOrderedAscending){

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

        localNotification.fireDate = self.dueDate;
        localNotification.timeZone = [NSTimeZone defaultTimeZone];
```

```
        localNotification.alertBody = self.text;
        localNotification.soundName = UILocalNotificationDefaultSoundName;

        localNotification.userInfo = @{@"ItemID" : @(self.itemId)};

        [[UIApplication sharedApplication]scheduleLocalNotification:localNotification];

        NSLog(@"Scheduled notification %@ for itemId %ld",localNotification,(long)self.itemId);

    }

}
```

上面的代码之前曾经学习过。使用上面的代码可以创建一个新的UILocalNotification对象。不过这里用到的是ChecklistItem的dueDate和text属性。此外我们还添加了一个userInfo词典，并将待办事项的ID作为唯一内容。通过这个词典我们才能在需要的时候找到该消息通知对象。

编译运行应用，添加一个新的checklistitem，然后设置截止日期为1分钟后。触碰done按钮，然后在Simulator中选择Hardware-Home。

端起咖啡或者茶静候一分钟，共同见证奇迹出现的那一刻。



虽然在日期选择器里面没有显示秒针读数，但他们实际上是存在的（只需要看看debug调试区的信息就知道了）。如果我们将截止日期设置为10：16 PM，而当前时间是10:15:54PM,那么久需要等到10:16:54。
如果我们可以将秒针读数设置为0就更好了，不过这里暂且不提。

好了，以上操作已经解决了一种情况，也就是创建新代办事项的时候自动添加一个新的本地消息通知。接下来还有两种情况需要处理：当用户编辑现有事项时，以及当用户删除一个事项时。

让我们先处理编辑代办事项的情况。

当用户编辑一个待办事项时，会发生以下可能：

1.Remind Me开关从关闭调整为开启状态，此时我们需要安排一个消息通知

2.Remind Me从开启调整为关闭状态，此时我们需要取消一个已有的消息通知

3.Remind Me仍然开启，但截止日期更改了。此时我们需要取消已存在的消息通知，然后安排一个新的。

4.Remind Me仍然开启，但截止日期没有发生变化。我们无需进行任何处理

5.Remind Me保持关闭，此时我们也无需做任何事情。

当然，在以上所有的情况下，只有当截止日期设置为未来才需要安排消息通知。

虽然已经是最后一章了，还是要废话重提一下。写代码之前先想一想该怎么做。你是人，不是机器。这个星球上的码农和码奴丧尸太多，会思考的真正人类太少。

看起来似乎我们要写一大堆逻辑代码来处理以上所有的情况，但实际上却没那么复杂。
首先我们要检查对于当前的待办事项是否有一个消息通知。如果有，那么先取消它。然后我们需要判断该事项是否应该有一个消息通知，如果是，就为它安排一个。
看起来这种算法很粗糙，不过挺管用。

首先在ChecklistItem.m中更改scheduleNotification方法：

```
-(void)scheduleNotification{

    UILocalNotification *existingNotification = [self notificationForThisItem];

    if(existingNotification !=nil){
        NSLog(@"Found an exisint notification %@",existingNotification);

        [[UIApplication sharedApplication]cancelLocalNotification:existingNotification];
    }

    if(self.shouldRemind &&
       [self.dueDate compare:[NSDate date]] != NSOrderedAscending){

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

        localNotification.fireDate = self.dueDate;
```

```
        localNotification.timeZone = [NSTimeZone defaultTimeZone];
        localNotification.alertBody = self.text;
        localNotification.soundName = UILocalNotificationDefaultSoundName;

        localNotification.userInfo = @{@"ItemID" : @(self.itemId)};

        [[UIApplication sharedApplication]scheduleLocalNotification:localNotification];

        NSLog(@"Scheduled notification %@ for itemId %ld",localNotification,(long)self.itemId);

    }

}
```

只需要添加黄色高亮部分的代码就可以了。
这里调用了一个新的方法notificationForThisItem，我们很快就会添加它的实现代码。如果该方法返回一个有效的UILocalNotification对象（非nil），那么在debug调试区输出一些信息，并让UIApplication对象取消这个消息通知。

接下来在ChecklistItem.m中添加一个新的notificationForThisItem方法：

```
-(UILocalNotification*)notificationForThisItem{

    NSArray *allNotifications = [[UIApplication sharedApplication]scheduledLocalNotifications];

    for(UILocalNotification *notification in allNotifications){

        NSNumber *number = [notification.userInfo objectForKey:@"ItemID"];

        if(number != nil &&[number integerValue] == self.itemId){
            return notification;
        }
    }
    return nil;


}
```

这里让UIApplication对象获取了所有已安排的消息通知。接下来使用for循环来遍历所有的消息通知，如果userInfo词典中的ItemID值和self.itemID属性相同，那么就找到了对应该Checklistitem的消息通知。如果没有相匹配的，该方法就返回nil。

这也是我们在编程中常用的一种模式。首先获取一堆项目的数组，然后遍历这个数组找到满足特定条件的项目，这里的条件是ItemID。一旦找到，就可以结束循环了。

编译运行应用，然后尝试添加一个代办事项，将截止日期设置为未来的几天后。此时会计划一个新的消息通知。然后编辑这个代办事项，并更改截止日期。旧的消息通知对象会被删除，并安排一个新的消息通知。我们可以从debug调试区的NSLog()输出中看到这一改变。

再次编辑这个代办事项，但是将Remind Me开关设置为OFF。旧的消息通知会被删除，同时不会添加新的消息通知。再次编辑，但是不要做任何更改，此时不会生成新的消息通知，因为开关仍然关闭。即便我们强关应用也不会影响这一点。

好了，还有最后一种情况需要处理：当删除某个ChecklistItem对象时。
而这最后一种情况又可能由两种操作导致：
1.用户使用swipe-to-delete滑动删除了一个单独的代办事项。
2.用户完全删除了一个cheklist列表。

在iOS中，当某个对象将被dealloc消息来删除的时候会得到通知。我们可以实现该方法，然后看看对该代办事项是否有一个消息通知，然后取消它即可。

在ChecklistItem.m的底部添加如下方法：

```
-(void)dealloc{

    UILocalNotification *existingNotification = [self notificationForThisItem];
    if(existingNotification !=nil){
        NSLog(@"Removing exisint notification %@",existingNotification);

        [[UIApplication sharedApplication]cancelLocalNotification:existingNotification];

    }
}
```

看，就是这么简单。

当用户删除某个单独的ChecklistItem代办事项，或者删除整个的Checklist都会调用dealloc方法。因为在删除整个Checlist的时候，属于它的所有ChecklistItem也会被删除。

编译运行应用。

首先对待办事项设置一些会在将来触发的消息通知（避免在你测试的时候就发送），然后删除该代办事项或是它所属的checklist。此时我们将在debug调试区中看到相关的消息。

好了，当我们可以确认一切工作正常的时候，就可以删掉NSLog()语句了。
此类语句只是为了调试目的才提供的。在最终上传的应用最好不要留下此类代码（尽管不会有太大影响）。

当然，别忘了从ChecklistViewController中删除标签的ID，因为它们也是仅用作调试母目的。

小练习1：

把截止日期放到表视图待办事项文本下面的table view cell中。

小练习2：
根据截止日期对待办事项进行排序。你可以参考之前对Checklist的排序，只是这里需要使用NSDate对象（NSDate对象没有localizedStandardCompare方法，但是有一个常规的compare方法）。

结束前的告白

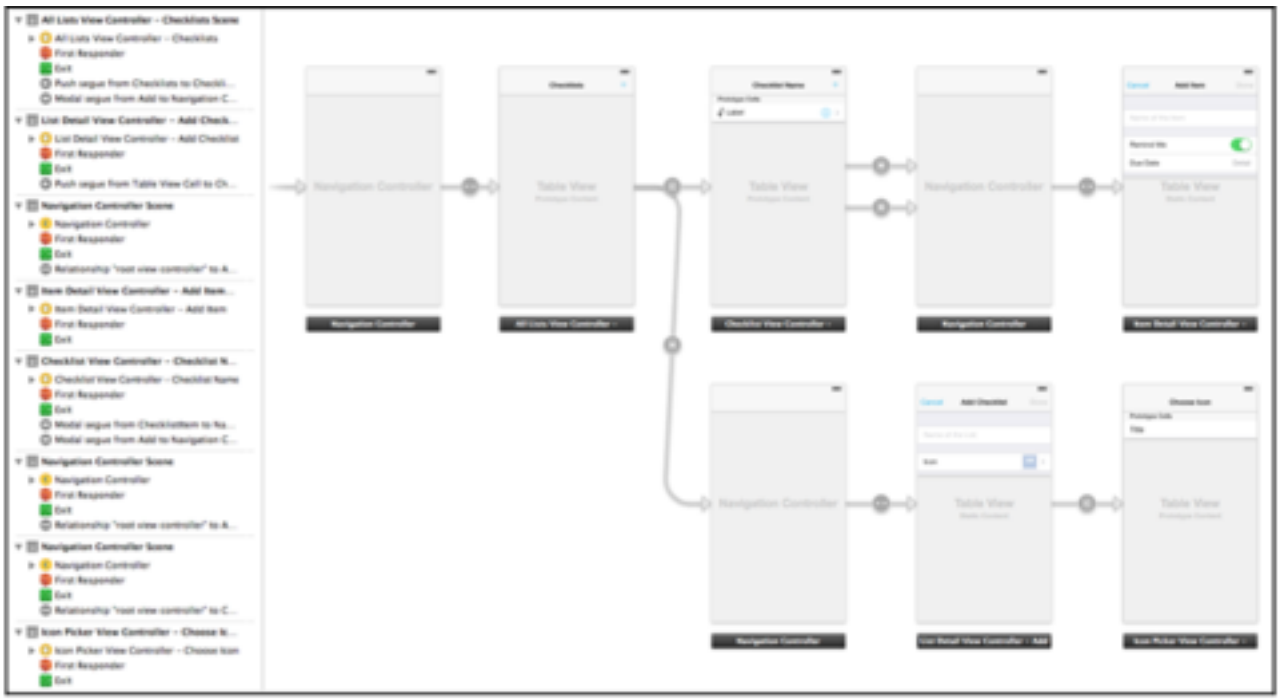
这一系列的教程到此算是正式结束了!

是的，没有 next chapter, 没有next big thing,更没有one more thing…

我们在这一系列的教程中学到的东西很多很多，或许你还没办法一次就完全看懂，但是，Don't Panic!
学习编程最重要的是学会一种思维模式，对于一种全新思维模式的建立，不要指望24小时可以搞定。

当然，本系列教程集中在UIKit上，而在下一系列的教程中，我们会回过头研究下Objective-C语言本身，并在最终完成的时候创建下一个NB的应用。

对于本系列教程，最终的storyboard是下面这样的：



好了，即便是告别，也还是要送上最后的福利~