从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter20

欢迎继续我们的学习。

开发环境：
Xcode 5 +iOS 7

经过这么长时间的努力，我们的应用正日趋完善。现在我们可以用它来创建checklists，并在其中创建待办事项。所有的数据信息都将保存在沙盒文件中，即便应用被强行关闭也不会受到影响。不过我们还需要对用户界面多一些改进工作。

假设用户正在查看Birthdays checklist的信息，然后突然想去京东商城看看礼品的价格信息，于是就按Home键切换到京东应用。此时Checklists应用就处于后台悬停状态。再假定因为某种原因Checklists应用被系统强行关闭了。当用户重新打开应用的时候，就会直接进入主界面，而不是刚才的Birthdays checklists界面。

对此你可能觉得没什么大不了，首先应用并不会经常被强行关闭（除非用户喜欢玩一些耗费内存的游戏），其次用户或许不在乎这个。不过请记住，在iOS应用开发的过程中，用户体验非常重要，哪怕是一个小小的不爽也会大大影响用户体验。对于用户来说，退出应用之前在某个界面，那么重新打开后还是希望在同一个界面。

遗憾的是，我个人试用过很多款iOS应用，包括很多巨头出品的应用，在这个小小的用户体验问题上却毫不在意。

我们可以将此类信息保存在Checklists.plist文件中，但是对于这种简单的设置信息，更方便的方法是借助NSUserDefaults对象的力量。

NSUserDefaults的工作原理类似词典，它是一个集合类对象，可以用来保存键值对。之前我们了解过数组这个集合类对象，它可以保存有序排列的对象。而词典则是iOS开发中另一种常用的集合类对象。

上图就是一个词典对象的例子，相信大家当年都用过新华字典或者英语词典。无论是字典还是词典，都是一个词条对应关于该词条的解释。
在Objective-C中，词典对象是由NSDictionary和NSMutableDictionary对象实现的。我们可以将对象保存在词典对象中，并提供一个引用的键。当后续需要使用它的时候，只需要使用这个键就可以了。事实上Info.plist文件的工作原理也是一样的。在实际使用的时候，程序会将plist文件读取到词典对象中，然后iOS根据不同的键（左侧红色部分）来获取不同的值（右侧部分）。通常这些键是以字符串的形式保存的，而值则可能是任何类型的对象。

NSUserDefaults实际上还不算是一种真正的词典对象，但它的工作原理也非常类似。当我们往NSUserDefaults中插入新的数值时，它们会被保存到应用的沙盒环境中，这样即便应用被强行关闭，相关数据信息仍然得以保存。

对于大量的数据信息，我们不建议使用NSUserDefaults，不过对于一些简单的设置信息（比如单机休闲游戏的关卡设置），NSUserDefaults是一个比较理想的场所。比如这里我们可以用它来保存上次应用退出时所在的界面。

那么具体该如何操作呢?

1.在从主界面（AllListsViewController)到checklist界面(ChecklistViewController)的segue上，我们需要编写代码将选中的checklist的行编号保存到NSUserDefaults中。这样我们就可以让应用记住用户所选择的checklist了。当然，或许你觉得保存checklist的名称比行编号更人性化。不过想想一种可能，如果两个checklists的名称相同肿么办？虽然这种情况比较罕见，不过用户不是完全例理性的，总有这种可能会出现。因此使用行编号就可以确保应用总是会选择正确的checlist

2.当用户触碰back按钮返回主界面的时候，我们需要从NSUserDefaults设置中删除刚才所保存的数值。通常情况下，当我们将某个数值设置为-1时，就意味着"没有数值"。为神马是-1？因为我们在计算行编号的时候是从0开始的，因此总是会使用0或者任何一个整数。-1不是一个有效的行编号，而因为它是一个负值，所以就很容易在debug的过程中观察到可能出现的情况。

3.当应用启动时如果NSUserDefaults中保存的数值不是-1,那么我们就知道用户之前在查看checklist中的内容，此时就可以手动执行segue切换到对应行的ChecklistViewController>

好吧，现在思路已经很清晰了。不过在实际写代码的时候，你会发现代码比起文字解释要简单的多。

在Xcode中切换到AllListsViewController.m，然后更改didSelectRowAtIndexPath的方法内容如下:

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
 [[NSUserDefaults standardUserDefaults]setInteger:indexPath.row forKey:@"ChecklistIndex"];

    Checklist *checklist = self.dataModel.lists[indexPath.row];

  [self performSegueWithIdentifier:@"ShowChecklist" sender:checklist];
}
```

上面黄色高亮的部分就是我们新添加的内容。

通过这一行代码，我们将所选择行的编号保存到NSUserDefaults中，其对应的键是"ChecklistIndex"。

当然，为了判断用户是否触碰了导航栏上的back按钮，我们需要为导航控制器设置一个代理对象。作为导航控制器的代理对象，就会从导航控制器那里了解到它何时将视图控制器push或pop到导航视图堆栈上。AllListsViewController是该代理的合适人选。

在Xcode中切换到AllListsViewController.h，然后在@interface这一行添加遵循代理协议声明：

```
@interface AllListsViewController : UITableViewController
<ListDetailViewControllerDelegate,UINavigationControllerDelegate>
```

看到这里千万不要害怕，一个视图控制器可以同时是多个对象的代理对象。比如这里的AllListsViewController既是ListDetailViewController的代理对象，也是UINavigationController的代理对象，同时还是UITableView的代理对象（不需要公开声明，因为它本身就是一个表视图控制器）。

接下来在AllListsViewController.m中添加代理方法的实现代码：

```
-(void)navigationController:(UINavigationController *)navigationController
willShowViewController:(UIViewController *)viewController animated:(BOOL)animated{

    if(viewController ==self){

        [[NSUserDefaults standardUserDefaults]setInteger:-1 forKey:@"ChecklistIndex"];
    }

}
```

每当导航控制器让应用切换到一个新的界面时都会调用该方法。如果用户触碰了back按钮，那么新的视图控制器就是AllListsViewController自身，此时我们可以设置NSUserDefaults中的ChecklistIndex值为-1，也就意味着没有选择任何的checklist

好了，还剩最后一步。当应用启动的时候，要确认需要显示哪个checklist，然后手动触发segue。

这个工作我们会放在viewDidAppear中进行。

在AllListsViewController.m中添加一个viewDidAppear方法：

```
-(void)viewDidAppear:(BOOL)animated{
    [super viewDidAppear:animated];

    self.navigationController.delegate = self;

    NSInteger index =[[NSUserDefaults standardUserDefaults]integerForKey:@"ChecklistIndex"];

    if(index != -1){

        Checklist *checklist = self.dataModel.lists[index];
```

```
        [self performSegueWithIdentifier:@"ShowChecklist" sender:checklist];
    }
}
```

当视图控制器开始呈现出界面的时候就会调用该方法。

首先，视图控制器(AllListsViewController)会设置其自身为导航控制器的代理对象。
然后会从NSUserDefaults设置中获取ChecklistIndex的值，将其保存到一个本地变量中。
接着根据所获取的行编号值进行判断，如果该值不是-1，就说明用户之前正在查看某个checlist，于是就手动触发segue并切换到对应的界面。
和之前一样，我们将Checklist对象放到sender的参数中，然后调用performSegueWithIdentifier切换界面。

注意!=的意思是不等于，它是==操作符的反义。有的编程语言会使用<>，不过在Objective-C中是无效的。

我的忏悔

写到这儿，我心中稍有些歉疚。viewDidAppear并非当应用启动的时候会调用，而是每次当导航控制器切换回主界面的时候都会调用。既然之前我们说过只需要在应用启动的时候确认是否需要恢复用户之前打开的checklist界面，那么为什么又要把这部分逻辑代码放在viewDidAppear中呢?

当AllListsViewController界面首次出现的时候，我们并不想调用willShowViewController这个代理方法，因为它会用-1覆盖ChecklistIndex的值，这样你就没机会恢复原来的界面了。通过将AllListsViewController设置成导航控制器的代理对象（直到它可见的时候），就可以成功避免这个问题。

当用户触碰back按钮的时候，导航控制器会在调用viewDidAppear方法前调用willShowViewController方法。因为"ChecklistIndex"的值此时将始终是-1,viewDidAppear方法就不会再触发segue了。

为了解决这个特殊情况，实际上有很多种方法，不过这里所采用的是比较简单易行的一种。

好吧，我猜这种逻辑已经让你头大了。别太担心。很可能第一遍你看不懂为什么要这么做，不过Don't Panic。你可以尝试着修改代码，然后看会发生些什么。在尝试了不同的方法之后，你才会明白为什么这么做是可行的。

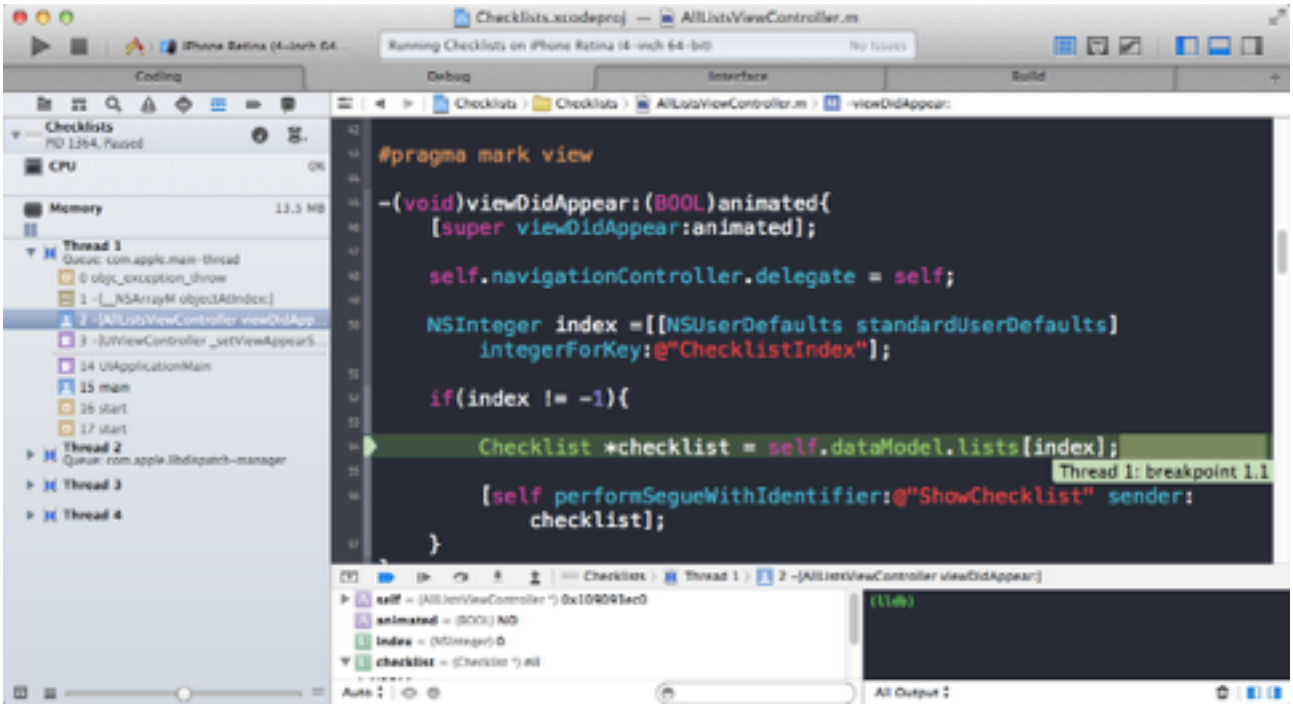
编译运行应用，然后切换到某个checklist界面。从simulator的菜单中选择Hardware-Home键，然后按Xcode中的Stop键来退出应用。

注意:

记得在Simulator上测试的时候，一定要先按Hardware-Home，因为NSUserDefaults需要时间将设置保存到硬盘中。如果你直接从Xcode中停止应用，就会丢失所做的保存。

好了，再次运行该应用，你会看到Xcode迅速切换回你之间所在的checklist界面。

接下来继续下面的工作:
在Xcode中Stop应用，然后从Simulator中删除该应用，或者选择iOS Simulator- Reset Contents and Settings删除。
然后编译运行应用，会看到它迅速崩溃了。



应用崩溃的位置在viewDidAppear方法的这一行：
Checklist *checklist = self.dataModel.lists[index];

好吧，究竟是神马状况？显然此时index变量的值是0。因为现在是初次安装应用，所以在NSUserDefaults中还没有任何内容，当然也没有给ChecklistIndex键赋值。

如果没有获取指定的键值，默认情况下NSUserDefault的integerForKey方法会返回0，但是在这个应用中0是一个有效的行编号。遗憾的是此时应用中还没有任何checklist，所以index 0在数组中并不存在。于是应用就会崩溃。

那么我们想要的是神马呢？我们希望当"ChecklistIndex"这个键在没有设置值的时候返回-1，因为对当前的应用来说，-1意味着显示主界面，而不是某个具体的checklist。幸运的是，在iOS中NSUserDefaults允许设置默认值。

在Xcode中切换到DataModel.m

然后在init方法之前添加一个新方法：

```
-(void)registerDefaults{

    NSDictionary *dictionary = @{@"ChecklistIndex" :@-1};

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

在上面的方法中，我们创建了一个新的NSDictionary对象，然后为ChecklistIndex这个键添加了值-1。当我们还没有设置数值的时候，NSUserDefaults会返回这个默认值。

接下来在DataModel.m中的init方法中添加代码调用这个新方法：

```
-(id)init{

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

        [self loadChecklists];
        [self registerDefaults];
    }
    return self;
}
```

再次编译运行应用，整个世界清静了~

为什么要在DataModel里面完成这个工作呢？好吧，我个人不喜欢看到和NSUserDefaults相关的代码在整个程序中到处都是。
接下来让我们来清理下代码，极爱那个所有和NSUserDefaults相关的代码放到DataModel里面去。

在DataModel.m中添加以下方法：

```
-(NSInteger)indexOfSelectedChecklist{

    return [[NSUserDefaults standardUserDefaults]integerForKey:@"ChecklistIndex"];
}

-(void)setIndexOfSelectedChecklist:(NSInteger)index{

    [[NSUserDefaults standardUserDefaults]setInteger:index forKey:@"ChecklistIndex"];

}
```

这样一来，剩下的代码就无需为NSUserDefault担心了。其它对象只需要在适当的位置调用DataModel的相关方法就可以了。

在面向对象编程中，向其它对象隐藏自己的实现细节是很重要的原则。如果我们后面打算使用其它方式来保存设置信息，比如保存到数据库中，那么只需要在DataModel中修改就好了。其它地方的代码不需要知道这些变化。

不过显然我们需要在DataModel.h中添加这两个方法的声明，否则其它对象将无法使用它们。

切换到DataModel.h，添加方法声明：

```
-(NSInteger)indexOfSelectedChecklist;

-(void)setIndexOfSelectedChecklist:(NSInteger)index;
```

接下来切换到AllListsViewController.m，更改以下方法的代码：

```
-(void)viewDidAppear:(BOOL)animated{
    [super viewDidAppear:animated];
```

```
    self.navigationController.delegate = self;

    NSInteger index = [self.dataModel indexOfSelectedChecklist];

    if(index != -1){

        Checklist *checklist = self.dataModel.lists[index];

        [self performSegueWithIdentifier:@"ShowChecklist" sender:checklist];
    }
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{

    [self.dataModel setIndexOfSelectedChecklist:indexPath.row];
    Checklist *checklist = self.dataModel.lists[indexPath.row];

  [self performSegueWithIdentifier:@"ShowChecklist" sender:checklist];
}


-(void)navigationController:(UINavigationController *)navigationController
willShowViewController:(UIViewController *)viewController animated:(BOOL)animated{

    if(viewController ==self){

        [self.dataModel setIndexOfSelectedChecklist:-1];
    }

}
```

再次编译运行应用，确保一切进展顺利。

现在应用可以清楚的知道你之前所打开过的界面，不过这个功能同时造成了一个难以被发觉的bug。

下面是重现这个bug的方式：
1.启动应用，
2.添加一个新的checklist，
3.在里面添加一个新的待办事项。
4.从Xcode直接强行关闭应用。

因为我们这里没有用Hardware-Home键的方式来关闭，因此新的checklist和事项并没有保存到Checklists.plist。不过在极少数情况下NSUserDefaults有可能已经将其设置保存到硬盘上，然后认为现在选中了新的列表。遗憾的是它现在根本不存在（没有保存到Checklists.list中）

NSUserDefaults会不时的将其信息变化保存，因此在你强关应用前，很可能信息已经被保存了。

编译运行应用试试看?
假如你运气足够好，很可能会看到应用崩溃。当然，大多数情况下是正常的。

问题的根源在于NSUserDefaults和Checklists.plist中的信息不同步。NSUserDefaults认为应用需要选中一个本不存在的checklist。

当然，这个情况是我们人为造成的。因为大多数情况下用户会使用Home键来退出应用，这样应用会进入后台悬停状态，因此Checklists.plist和NSUserDefaults都会得以保存，所有数据信息都是同步的。不过iOS会在内存不足或某些特定情况下强关应用，此时应用就会崩溃。显然这会给用户体验带来极大的伤害。

永远记住iOS设计开发的第一要义，用户体验至上。即便这种bug发生的可能性很小，不过我们仍然要采取措施避免它的出现。这就是所谓的defensive programming（防御型编程），总是检查一些边界特例情况，然后确保它们不会破坏用户体验。

切换到AllListsViewController.m，然后更改viewDidAppear方法如下：

```
-(void)viewDidAppear:(BOOL)animated{
   [super viewDidAppear:animated];

   self.navigationController.delegate = self;

   NSInteger index = [self.dataModel indexOfSelectedChecklist];

   if(index >=0 && index <[self.dataModel.lists count]){

      Checklist *checklist = self.dataModel.lists[index];

      [self performSegueWithIdentifier:@"ShowChecklist" sender:checklist];
   }
}
```

这里我们不光检查index != -1，而且要检查index编号是否是有效的。如果它在0和数据模型的checklist数量之间，那么就是有效的，否则就无需触发segue。这样就会避免self.dataModel.lists[index]获得某个根本不存在的对象。

好吧，这里又出现了一个新的操作符&&. 它的含义是逻辑与。通常我们这样来用它：

```
if (something && somethingElse) {
// do stuff
}
```

它的意思是：如果something是true，而且somethingElse也是true,那么就开干。

在viewDidAppear中，仅当index编号大于等于0，而且还小于checklist的数量时，才会触发segue。

有了这个防御型编程的检查工作，现在应用就不会意外触发某个不存在的checklist，即便数据部同步也不会。
记住，应用并不知道用户是否打开了Add/Edit Checklist或Add/Edit item的界面。这种modal界面属于临时性界面。我们通过此类界面来更改一些信息，然后将其关闭。当应用进入后台或被强关的时候，这类modal界面是否消失并不重要。
好吧，至少对于我们这款应用应该是这样的。不过如果你在应用中允许用户在modal界面中做出很多复杂的编辑工作，可能也希望可以保存这些变化。
在这篇教程中我们使用NSUserDefaults来保存界面开启的信息，不过iOS实际上提供了其它的API更加胜任此类工作，State Preservation和Restoration。这里就先不赘述了。
好了，今天的内容稍微有点长，不过我们认识了iOS开发中又一个非常重要的类，NSUserDefaults。

今天还是：Merry Christmas ,and a happy new year!


Photography