从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter19

欢迎继续我们的学习。

开发环境：
Xcode 5 +iOS 7

如果你一路看到现在，那么恭喜你，在一周左右的时间内我们就会圆满完成这一系列教程的学习了。

到目前为止，数据模型的表现还可以接受，不过我们还可以再优化一点。我们已经为Checklist何ChecklistItem创建了数据模型对象，但在AllListsViewController中仍然有用于保存Checklists.plist文件的代码，实际上这本应该也是属于数据模型的工作。

在我开发iOS应用的过程中，我个人偏好设计自顶向下的数据模型对象。比如对这个应用来说，数据模型对象将包含Checklist对象，我们可以将加载和保存数据的代码都转移到新的数据模型。

首先让我们往项目中添加一个新的文件，选择Objective-C class模板，设置subclass of NSObject，然后将其命名为DataModel。

在Xcode中切换到刚刚创建的DataModel.h，然后更改其中的代码如下：

```
#import <Foundation/Foundation.h>

@interface DataModel : NSObject

@property(nonatomic,strong)NSMutableArray *lists;

-(void)saveChecklists;

@end
```

在上面的代码中我们添加了一个lists属性变量声明，以及一个saveChecklist方法的声明。
DataModel希望可以接管当前由AllListsViewController所负责的加载、保存数据信息的工作。

切换到DataModel.m，添加以下方法：

```
-(NSString*)documentsDirectory{

   NSArray * paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
   NSString *documentsDirectory = [paths firstObject];

   return documentsDirectory;

}
```

```
-(NSString*)dataFilePath{

    return [[self documentsDirectory]stringByAppendingPathComponent:@"Checklists.plist"];
}


-(void)saveChecklists{

    NSMutableData *data = [[NSMutableData alloc]init];
    NSKeyedArchiver *archiver = [[NSKeyedArchiver alloc]initForWritingWithMutableData:data];
    [archiver encodeObject:self.lists forKey:@"Checklists"];
    [archiver finishEncoding];
    [data writeToFile:[self dataFilePath] atomically:YES];

}

-(void)loadChecklists{

    NSString *path = [self dataFilePath];
    if([[NSFileManager defaultManager]fileExistsAtPath:path]){
        NSData *data =[[NSData alloc]initWithContentsOfFile:path];
        NSKeyedUnarchiver *unarchiver = [[NSKeyedUnarchiver
alloc]initForReadingWithData:data];

        self.lists = [unarchiver decodeObjectForKey:@"Checklists"];

        [unarchiver finishDecoding];

    }else{

        self.lists = [[NSMutableArray alloc]initWithCapacity:20];
    }

}
```

以上方法的内容都是我从AllListsViewController.m中剪切粘贴过来的。如果你不是这样做的，那么需要切换到AllListsViewController.m，然后删掉以上方法的代码。
不过注意到上面用黄色高亮的部分是和原方法代码稍有差异的地方，因为array数组现在是属性变量，因此我们用self.lists替代了_lists。

别忘了从AllListsViewController.m中删除initWithCoder方法。
然后在DataModel.m中添加如下的init方法。

```
-(id)init{

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

        [self loadChecklists];
    }
```

```
    return self;
}
```
通过以上的方法可以确保当应用创建DataModel对象时就会加载Checklists.plist文件。

接下来更改AllListsViewController.h的代码如下：

```
#import <UIKit/UIKit.h>
#import "ListDetailViewController.h"

@class DataModel;

@interface AllListsViewController : UITableViewController <ListDetailViewControllerDelegate>


@property(nonatomic,strong)DataModel *dataModel;


@end
```

这里我们删除了之前的saveChecklists方法声明，然后添加了dataModel属性变量声明。

再次确认你已经从AllListsViewController.m中删除了
documentsDirectory,dataFilePath,saveChecklists,loadChecklists等方法。

接下来删除@implementation语句花括号中的_lists实例变量。

然后在文件顶部添加一行代码：
```
#import “DataModel.h"
```

好了，现在我们无需直接引用_lists变量，因为它根本就不村子。取而代之的是，我们将试图获取DataModel对象的lists属性。

在AllListsViewController.m中更改以下方法的代码：

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return [self.dataModel.lists count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
  static NSString *CellIdentifier = @"Cell";

  UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
  if (cell == nil) {
```

```
    cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier];
  }

  Checklist *checklist = self.dataModel.lists[indexPath.row];
  cell.textLabel.text = checklist.name;
  cell.accessoryType = UITableViewCellAccessoryDetailDisclosureButton;

  return cell;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
  Checklist *checklist = self.dataModel.lists[indexPath.row];

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

- (void)tableView:(UITableView *)tableView commitEditingStyle:
(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath *)indexPath
{
  [self.dataModel.lists removeObjectAtIndex:indexPath.row];

  NSArray *indexPaths = @[indexPath];
  [tableView deleteRowsAtIndexPaths:indexPaths
withRowAnimation:UITableViewRowAnimationAutomatic];
}


- (void)listDetailViewController:(ListDetailViewController *)controller didFinishAddingChecklist:
(Checklist *)checklist
{
  NSInteger newRowIndex = [self.dataModel.lists count];
  [self.dataModel.lists addObject:checklist];

  NSIndexPath *indexPath = [NSIndexPath indexPathForRow:newRowIndex inSection:0];
  NSArray *indexPaths = @[indexPath];
  [self.tableView insertRowsAtIndexPaths:indexPaths
withRowAnimation:UITableViewRowAnimationAutomatic];

  [self dismissViewControllerAnimated:YES completion:nil];
}

- (void)listDetailViewController:(ListDetailViewController *)controller didFinishEditingChecklist:
(Checklist *)checklist
{
  NSInteger index = [self.dataModel.lists indexOfObject:checklist];
  NSIndexPath *indexPath = [NSIndexPath indexPathForRow:index inSection:0];
```

```
  UITableViewCell *cell = [self.tableView cellForRowAtIndexPath:indexPath];
  cell.textLabel.text = checklist.name;

  [self dismissViewControllerAnimated:YES completion:nil];
}

- (void)tableView:(UITableView *)tableView accessoryButtonTappedForRowWithIndexPath:
(NSIndexPath *)indexPath
{
  UINavigationController *navigationController = [self.storyboard
instantiateViewControllerWithIdentifier:@"ListNavigationController"];

  ListDetailViewController *controller = (ListDetailViewController
*)navigationController.topViewController;
  controller.delegate = self;

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

  [self presentViewController:navigationController animated:YES completion:nil];
}

@end
```

注意上面黄色高亮显示的就是我们所做的更改，也即使用self.dataModel.lists取代了之前的_lists实例变量。
重复说明下我们刚才所做的事情，我们创建了一个新的DataModel对象，其中包含了Checklists对象数组，并且知道如何加载和保存checklist和其中的代办事项。此时AllListsViewController将不再使用自己的数组，而改为通过访问self.dataModel的属性来使用DataModel对象，

看起来有点意思，不过DataModel对象在哪里创建呢？当前的代码中并没有一处对DataModel对象初始化，比如类似[[DataModel alloc]init]。

显然，创建DataModel的最佳地点是在应用的app delegate中。我们可以将app delegate看作应用最顶层的对象。因此让它成为数据模型的主人似乎是比较恰当的。在创建了DataModel之后，app delegate接下来会将它移交给需要使用它的视图控制器。

在Xcode中切换到ChecklistsAppDelegate.m，然后在文件顶部添加以下代码：

```
#import “DataModel.h"
```

然后添加一个实例变量：

```
@implementation ChecklistsAppDelegate{
   DataModel *_dataModel;
}
```

接下来更改saveData方法为：

```
-(void)saveData{

    [_dataModel saveChecklists];
}
```

如果你此时迫不及待的开始编译运行应用，那么当你添加一个新的checklist时应用就会崩溃，因为_dataModel还是nil。创建DataModel实例的最佳地点是在application:didFinishLaunchingWithOptions:方法中，因为它是应用启动后将调用的首个方法之一：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    _dataModel = [[DataModel alloc]init];

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

    AllListsViewController *controller = navigationController.viewControllers[0];

    controller.dataModel = _dataModel;


    return YES;
}
```

在上面的代码中，首先会创建一个DataModel对象，然后将通过之前我们所介绍的方式在storyboard中找到AllListsViewController，并设置其dataModel属性。

现在从Xcode的菜单中选择Product -Clean，然后编译运行应用。
看看一切是否可以顺利工作?

好了，今天的学习内容相对比较轻松，就先到这里吧。

送上今日福利。
Merry Christmas and a happy new year!