从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter14

欢迎继续我们的学习。

开发环境：
Xcode 5.1 DP 2 +iOS 7.1 beta2

在上一章的内容中，我们成功学会了如何将代办事务清单保存到应用沙盒的文件中。不过只懂得保存还不够，接下来我们要学习如何加载Checklists.plist文件。其实很简单，你只需要将之前做的事情反过来就可以了。

还记得Xcode的那个警告吗？ChecklistItem.m还没有实现NSCoding协议的另外一个方法initWithCoder。这个方法将用于从文件中解冻对象。

在Xcode中切换到ChecklistItem.m，在encodeWithCoder方法之前添加以下方法：

```
-(id)initWithCoder:(NSCoder *)aDecoder{

   if((self =[super init]))
   {
      self.text = [aDecoder decodeObjectForKey:@"Text"];
      self.checked = [aDecoder decodeBoolForKey:@"Checked"];
   }
   return self;
}
```

在initWithCoder方法中，我们做了和encodeWithCoder方法相反的事情。我们从NSCoder的decoder对象中取出了对象，然后将值保存在ChecklistItem的属性中。好了，就这么简单。之前我们保存在”Text”这个key下面的值将赋予self.text属性，同样的，保存在”Checked”这个key下面的值将赋予self.checked属性。

至于加载Checklists.plist文件的工作将由NSKeyedUnarchiver对象完成。unarchiver将在幕后完成下面的工作从而创建ChecklistItem对象：

```
ChecklistItem *item = [[ChecklistItem alloc] initWithCoder:someDecoderObject];
```

但是当我们手动创建ChecklistItem的时候，会用下面的方式：

```
ChecklistItem *item = [[ChecklistItem alloc] init];
```

区别在于NSKeyedUnarchiver不再使用常规的init方法，而使用了initWithCoder方法。这一点其实我们应该习以为常了。在iOS开发中，对象通常有不止一个init初始化方法。当然，不管怎样，方法的名称总是”initSomething”，这是个命名习惯。至于使用哪个初始化方法则取决于具体的情况。

比如当用户触碰+按钮，并在Add Item界面中创建新的ChecklistItem对象时，我们会采用常规的init方法。而当我们从文件中加载ChecklistItems对象时则会使用initWithCoder方法。

理论知识充电时间到了。

关于init初始化方法

在Objective-C中，使用init开头的方法具备某种特殊的含义。如果在iOS开发中我们要给某个方法取名为initXXXX，那么这个方法就肩负着创建新对象的重任。当使用init开头的方法时，首先我们会给对象分配一个足够大的内存空间，用来保存对象中的数据（其实例变量），然后调用一个init方法来初始化对象，从而让其立即可用。

这些init初始化方法，无论是叫init还是initWithCoder，还是initXXXXX，总是按照这一步骤来创建对象。当我们编写自定义的init方法时，也必须遵循以上的步骤。

下面是编写一个init初始化方法的标准方式：

```
-(id)init
{
        self = [super init];
        if(self){
                //初始化代码
        }

}
```

首先我们会调用[super init]来初始化对象的父类，然后将其结果赋予self。好吧，假如你之前学过其它的编程语言，对这句代码肯定觉得很怪异。但很无奈，这就是Objective-C的方式。

假如你之前根本没学过其它面向对象的语言，那么superclass(父类）究竟是个神马东西？这个问题很好，不过现在我们无需在这个问题上深究（当然你可以去wiki或者百度百科上先了解个大概），在下一系列的教程中会涵盖相关内容。你只需要记住，有时对象需要给某个叫super的东西发送消息。而如果你忘了这么做，程序就会崩溃。

在将[super init]赋予self之后，我们会检查self的值，并判断它是否是nil

```
        if(self){
                //初始化代码
        }
```

这里的if(self)其实等同于if(self!=nil)

在ChecklistItem的initWithCoder方法中，我们使用了Objective-C的习惯写法：

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

}
```

以上代码其实是两行代码合并成一行的结果：

```
self = [super init];
if (self != nil) {
... }
```
如果你想让整个语句更加明确，还可以这样写：

```
if ((self = [super init]) != nil) { ...
}
```
不过最通常的写法还是：

```
if ((self = [super init])) { ...
}
```

一定要注意的是：
这里用了双重括号！！！！！！

如果你这样写：
```
if (self = [super init]) { ...
}
```

那么Xcode会提示出错。因为如果用上面的写法，Xcode就会被你弄糊涂了，究竟你是要给self赋值，还是要比较self和[super init]的返回值。
因此我们会使用双重括号确保编译器可以理解我们的意思。

以下写法都是可以的：

```
self = [super init]; if (self) { . . . }
self = [super init];
if (self != nil) { . . . }
if ((self = [super init])) { . . . }
if ((self = [super init]) != nil) { . . . }
```

不过我们还是推荐官方的写法：

```
if ((self = [super init])) { ...
}
```

存在着这样一种可能，也就是[super init]返回了nil，因为对于该对象父类的初始化失败。此时if语句的作用就体现出来了。

不过幸运的是，这一幕发生的概率非常之低。广州恒大的世界级教头里皮先生说过，
“足球场上永远无法预测会发生什么，与拜仁这样的顶尖球队交手100次可能会输掉99场，但有机会会赢下1场，没有人知道下一场比赛的结果是什么。”
实际上，[super init]返回nil的可能性或许是万分之一，甚至是百万分之一吧。

不过还是有发生的可能性，足球是圆的，代码是二进制的，一切皆有可能。
举个例子，假如对象在初始化的时候想要加载一个图片，但这个图片却不存在，而没有它我们的对象就无法工作。此时我们不得不销毁对象，然后返回nil。
比如UIImage对象使用一个图片文件来初始化对象的时候，但在application bundle里面却步存在这样一个图片，于是就会返回nil。

那么广州恒大战胜拜仁会在怎样的情况下发生呢？北大叫兽说的好，在梦中！

如果self不是nil，那么父类就已经成功初始化，而我们就可以执行当前对象的初始化工作了。例如，在ChecklistItem的initWithCoder方法中，我们通过从NSCoder对象中读取的值来初始化对象，然后将它们赋予ChecklistItem的属性。

好了，在初始化方法的最后我们会返回一个self.
这是必不可少的一步，唯有如此，调用方法达到对象才能将新对象分配给某个变量。

最后总结一下初始化方法的工作原理：

1.调用[super init]，并将结果赋予self
2.检查self是否是nil，如果是，就退出该方法，并返回nil
3.如果self不是nil,那么就执行后续的初始化工作。同样这意味着要赋予实例变量初始值。默认情况下，对象是nil,int变量是0,BOO变量是NO。而在初始化方法中我们将赋予它们不同的初始值。
4.将self返回给调用初始化方法的对象（或方法）。

当然，我们并非一定要手写一个init初始化方法。如果初始化方法什么也不做，也就是说不需要给属性或者实例变量赋值，那么可以不必管它。编译器会提供一个系统默认的init方法来保证程序的顺利运转。

这就是为什么我们可以直接调用[[ChecklistItem alloc]init]，而实际上在ChecklistItem.m中我们并没有提供一个名为init的方法（只有一个initWithCoder)方法。

好了，现在ChecklistItem的内容已经基本完整，现在它可以将之前被序列化（冷冻）到plist文件中的对象解冻了。不过我们还需要手动编写代码来加载plist文件，这个方法的内容将在ChecklistsViewController中添加。

到目前为止，我们都是在viewDidLoad方法中完成数据模型的初始化。对于本教程来说，不失为一种方便之举，但实际上不是合理的方式。viewDidLoad方法通常用于设置视图，比如应用中的可视化界面元素。

数据模型对象应更早进行初始化，因为通常情况下数据模型比视图的寿命周期更长。既然现在我们已经了解了init方法，那么更合理的做法就是在ChecklistsViewController的init初始化方法中创建数据模型。

不过需要注意的是，表视图控制器和其它对象一样有不止一种的init方法，比如：

initWithCoder，比如从storyboard中自动加载的视图控制器就会使用该初始化方法

initWithNibName，如果我们要手动从一个nib文件中加载视图控制器，就会使用该初始化方法。(nib文件其实和storyboard的作用相似，只是其中仅包含一个视图控制器）。

initWithStyle，如果我们不想通过storyboard或者nib文件来创建一个表视图控制器，就可以使用该初始化方法

在我们这个应用中，视图控制器加载自storyboard，因此我们会使用initWithCoder犯法来创建数据模型，并加载plist文件。是的，也就是我们刚刚在ChecklistItem中所实现的同名方法。UITableViewController对象会使用相同的NSCoder系统从storyboard文件中解冻加载。既然storyboard觉得它不错，那么我们也可以接受！

在Xcode中切换到ChecklistsViewController.m，然后在viewDidLoad方法前添加一个initWithCoder方法：

```
-(id)initWithCoder:(NSCoder *)aDecoder{
   if((self =[super initWithCoder:aDecoder])){
      [self loadChecklistItems];
   }
   return self;

}
```

注意这里初始化方法的标准模式：首先我们调用super，然后将结果赋予self。如果self 不等于nil,那么我们就开始具体的初始化工作，并最后返回一个到self的引用。区别在于这里不是调用[super init]，而是[super initWithCoder]。通过这种方式，可以确保视图控制器的其它部分可以从storyboard文件中顺利解冻。

接下来在initWithCoder：方法前添加loadChecklistItems的实现代码：

```
-(void)loadChecklistItems{

   NSString *path =[self dataFilePath];

   if([[NSFileManager defaultManager]fileExistsAtPath:path]){

      NSData *data = [[NSData alloc]initWithContentsOfFile:path
                  ];
      NSKeyedUnarchiver *unarchiver = [[NSKeyedUnarchiver
alloc]initForReadingWithData:data];
```

```
        _items = [unarchiver decodeObjectForKey:@"ChecklistItems"];
        [unarchiver finishDecoding];

    }else{

        _items = [[NSMutableArray alloc]initWithCapacity:20];
    }

}
```

让我们来看看以上方法的具体作用。

这个方法的主要内容可以分为两部分：

```
NSString *path = [self dataFilePath];
if ([[NSFileManager defaultManager] fileExistsAtPath:path]) {
...
} else {
_items = [[NSMutableArray alloc] initWithCapacity:20];
}
```

首先我们将[self dataFilePath]返回的结果（也就是应用沙盒）保存在一个名为path的临时变量中，接下来我们沙盒中是否存在该文件。
如果没有，那么显然就没有任何ChecklistItem对象可供加载。此时我们跳转到else部分，创建一个空的NSMutableArray。这也是我们在viewDidLoad方法所做的工作。
当应用从沙盒中找到Checklists.plist文件时，我们无需创建一个新的数组。此时我们可以从Checklists.plist文件中加载整个数组和其中的内容：

```
NSData *data = [[NSData alloc] initWithContentsOfFile:path];
NSKeyedUnarchiver *unarchiver = [[NSKeyedUnarchiver alloc]
initForReadingWithData:data];
_items = [unarchiver decodeObjectForKey:@"ChecklistItems"]; [unarchiver finishDecoding];
```

以上内容其实是对saveChecklistItems的逆向操作。首先我们将文件内容加载到一个NSData对象中。接着我们创建了一个NSKeyedUnarchiver对象（也就是unarchiver），然后让其将数据集解码到_items数组中。这样就创建了一个NSMutableArray，然后用ChecklistItems对象来填充其中的内容。

现在我们可以从viewDidLoad犯法中删掉之前的伪数据了：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
        // Do any additional setup after loading the view, typically from a nib.

}
```

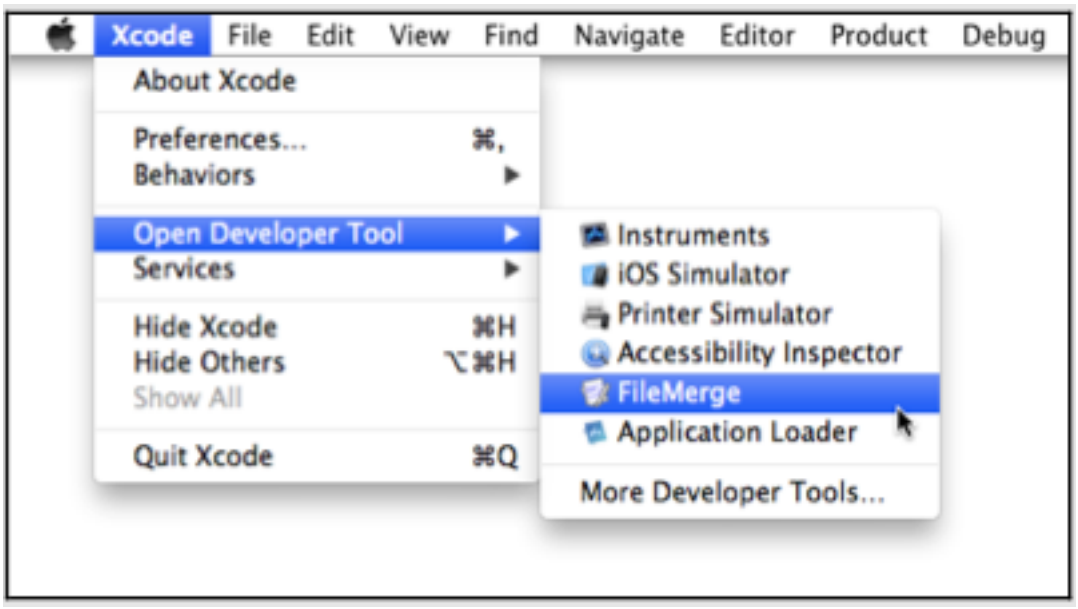
好了，现在编译运行应用。然后添加一些待办事项，用XXcode的Stop来关闭应用。再次打开看看所做的改变是否在其中。

为了进一步验证，我们从finder中进入沙盒的Documents文件夹，并删除其中的plist文件。再次编译运行应用。整个世界再次回到起始状态了。尝试添加一个项目，就会发现.plist文件再次重现。

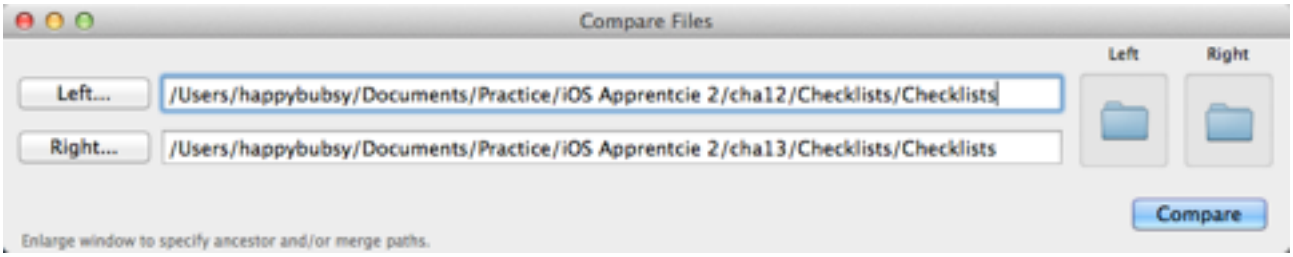
太棒了！我们再次攻克了一个新的难关！到目前为止，所创建的应用可以添加和编辑数据，而且还可以持久化保存数据。这些技术看起来很简单，但组合起来却可以构成很多复杂应用的根基。
想想看你已经学到了哪些知识？比如如何使用表视图控制器，如何使用导航控制器，如果使用storyboard的segue，如何使用代理传送数据，如何持久化保存数据。。。
所有的这些都是非常基础但又非常重要的iOS开发技巧！

在结束本章内容之前，我们再来学习一个Xcode的小技巧。
Xcode中提供了一个名为FileMerge的工具，可以用来比较文件内容。
比如你可以用FileMerge来比较自己版本的应用和我提供的示例代码之间的区别。

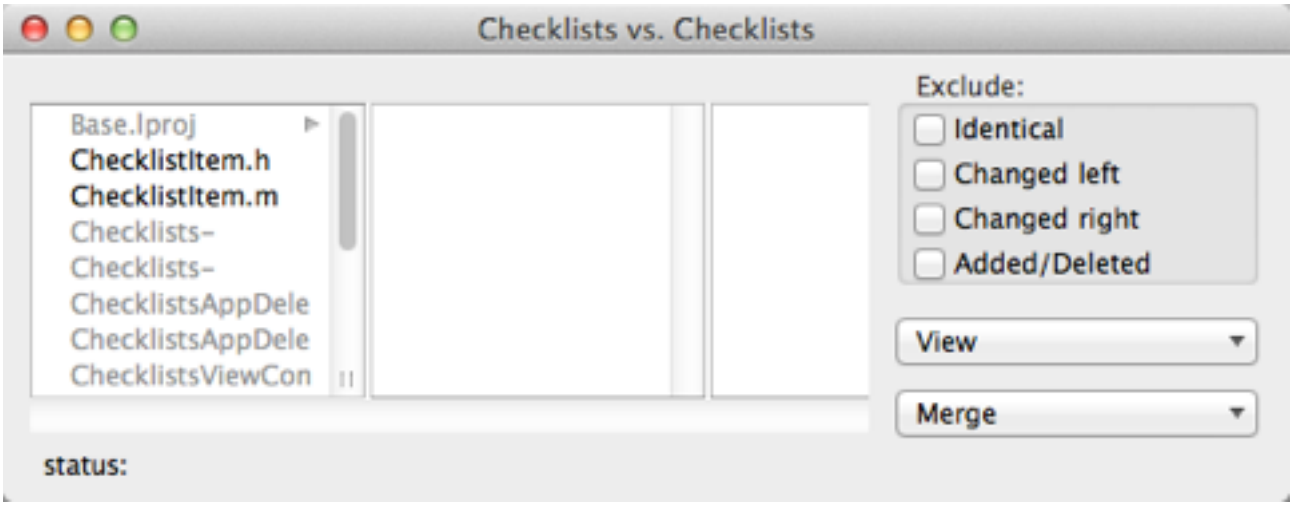
从Xcode的菜单打开Open Developer Tool- FileMerge



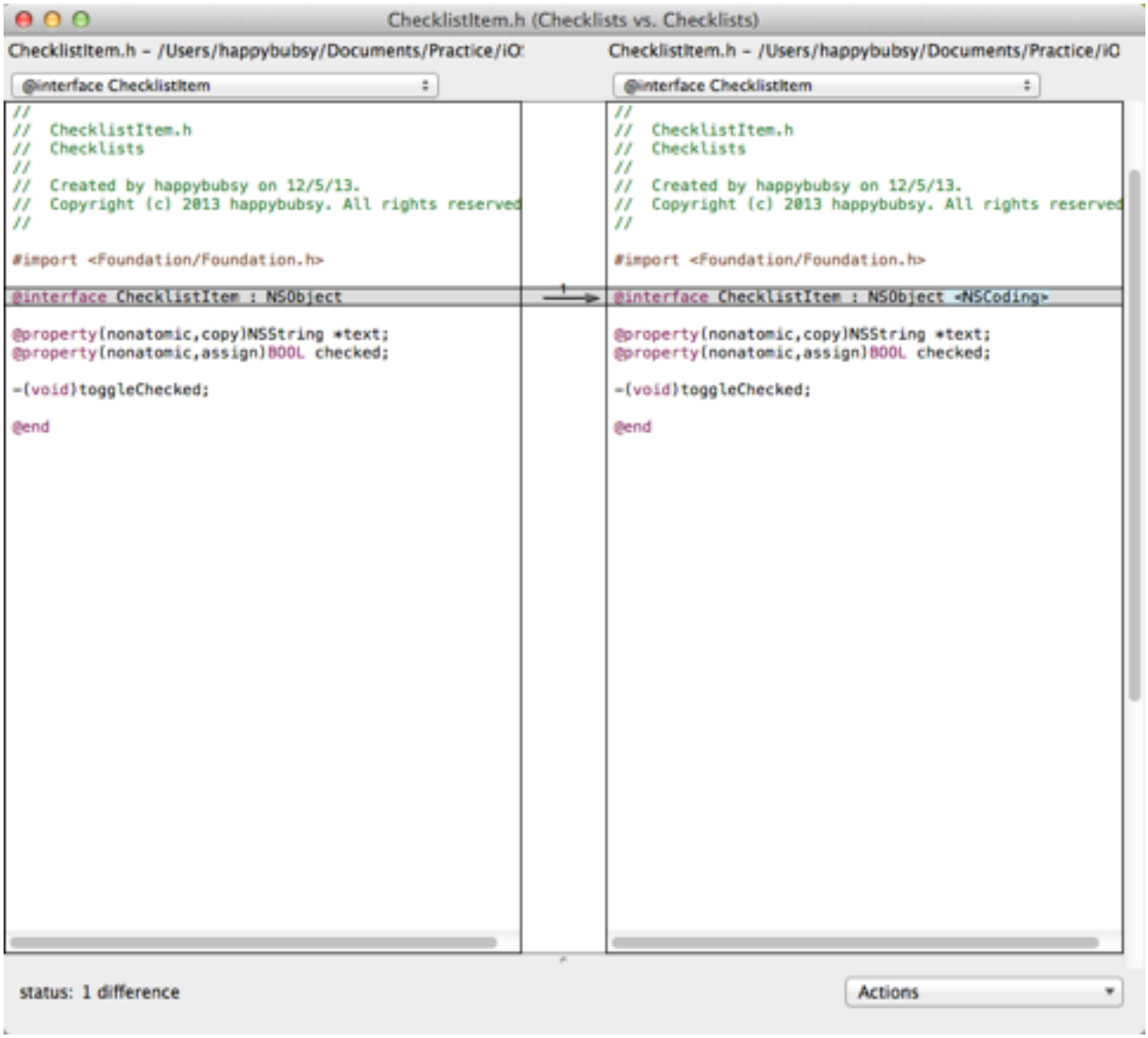
然后可以提供两个文件或者文件夹用于比较：



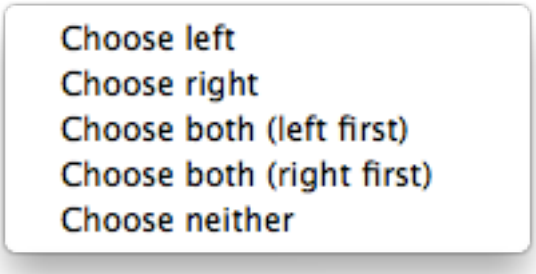
比如这里我们将cha12和cha13的代码进行比较，点击Compare，FileMerge会提示区别所在：



上面的黑色标出的文件名就是有区别的地方。
双击某个文件名可以看到具体的差异：



然后你可以从Actions里面选择操作：



当然这里我们选择什么也不做。

FileMerge对于初学者的最大好处是，当我们跟着某个示例教程学习的时候，如果发现代码出问题了，但死活找不到出错的地方，可以直接拿自己的项目和别人提供的示例代码进行比较。

好了，今天的学习到此为止，我们已经取得了阶段性的胜利，值得小小的庆祝一下~