从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter3

欢迎继续我们的学习。

首先还是要脑补一些理论知识。

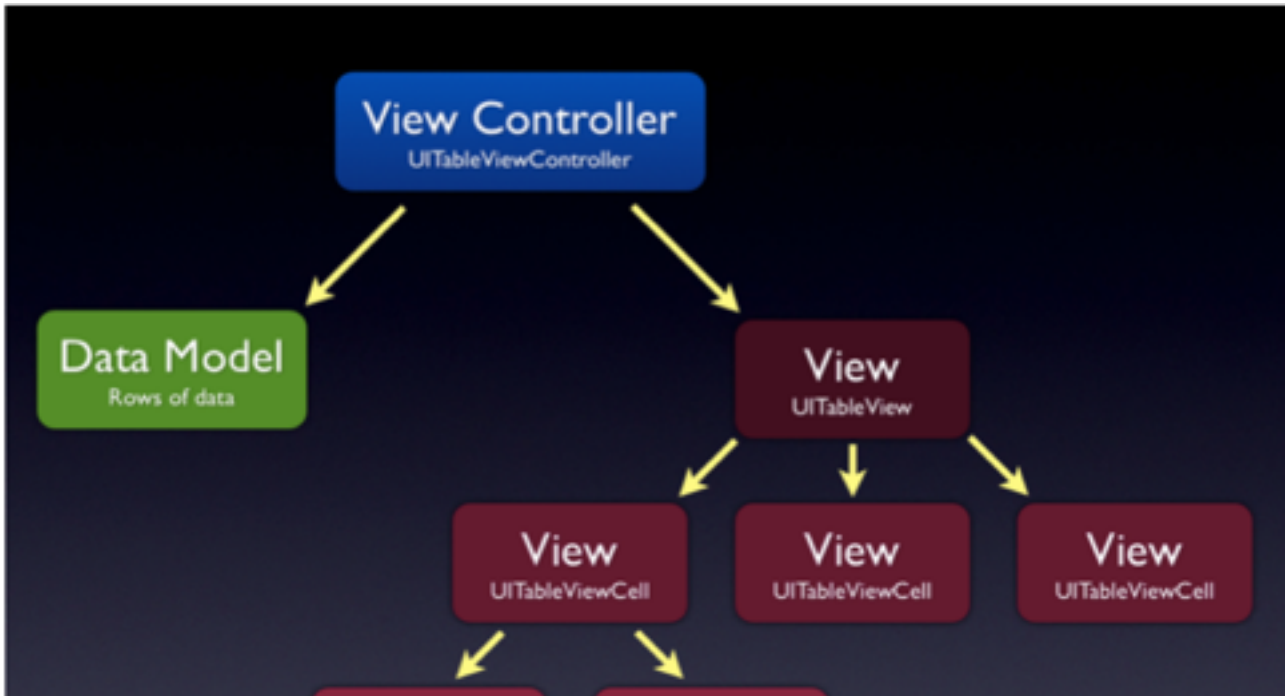
Model-View-Controller（模型-视图-控制器）

传说中NB无比的MVC再次出现了。任何一个iOS开发教程都不能掠过对Model-View-Controller，也就是MVC的解释说明。在iOS开发中，MVC是三大基础模式之一。实际上我们已经领教过另外两种模式的厉害了，它们分别是：delegation（代理/委托），也就是让某个对象代表另一个对象做一些事情。然后是target-action，通过这种方式可以让交互事件（比如触碰）和特定的动作方法关联在一起。
通过MVC模式，可以将应用中的所有对象分成三大类：
1.Model 对象，也就是数据模型。model对象中包含了应用中的数据以及对数据的任何处理。假如我们要开发一款菜谱应用，那么model对象显然就包含了所有的菜谱。在游戏中，model对象可能是不同管卡的设计，玩家的得分和怪物所在的位置。对数据模型对象的操作有时候被万恶的术语党称为"业务规则"或者"领域逻辑"，"业务逻辑"神马的。不要被这些名词吓倒，Don't panic! 不管这些名词怎么叫，你只需要知道它们的本质是关于如何处理数据就好了。
在我们这个事务管理软件应用中，事务清单和代办事务构成了数据模型。

2. View对象，也就是视图对象。视图对象其实就是我们应用中的所有可视元素，比如图片，按钮，标签，文本框等等。在游戏中视图对象就是游戏世界的视觉呈现，比如怪物攻击的动画，血条法条等等。一个视图对象可以绘制自身，同时对用户输入做出响应，但它通常不会直接处理应用中的逻辑。很多视图对象（比如UITableView)可以在不同类型的应用中重用，就是因为它们没有和某种特定的数据模型绑定在一起。

3.Controller对象。在iOS开发中将其称为视图控制器，它的作用是负责将数据模型对象和视图对象关联在一起。视图控制器会监听用户对视图的触碰事件，然后让数据模型对象做出响应的处理，并根据所处理的结果来更新视图的显示。视图控制器是MVC模式中的主宰力量。

下面的这个图显示了MVC中的不同类型对象是如何相同作用的



在iOS开发中，一个视图控制器会有一个自己的主视图，可以通过self.view属性来访问。主视图中包含了一系列的子视图。我们经常会看到一个界面中有多个子视图存在。处于视图层级顶端的视图通常会覆盖整个屏幕界面。而我们通常在storyboard（或nib文件）中来设计视图控制器所管理的界面的视觉元素布局。

在我们当前这款应用中，主视图就是UITableView，而它的子视图则是table view cell。每个cell也都有各自的子视图，比如文本标签，附属（如勾选标志）等等。

一个视图控制器只负责管理应用中的一个界面。如果应用中有多个界面，那么每个界面都有自己的视图，同时每个界面又被各自的视图控制器所管理。在界面跳转的时候，掌控权就会从一个视图控制器转交到下一个视图控制器。

在iOS开发的过程中，我们经常会需要创建属于自己的视图控制器。不过iOS中提供了一些方便易用的视图控制器，比如用于撰写邮件的mail compose 视图控制器，用于从照片库中选择照片的image picker 视图控制器，用于发送Twitter消息的tweet sheet，等等。

继续脑补。

关于视图和视图控制器的区别

相信在看了对MVC的解释后，大家对视图和视图控制器的区别应该略有所悟。不过这里还是要再次强调下，以便加深印象。在iOS的程序世界中，视图和视图控制器是两个完全不同的生物。视图对象用于在屏幕界面上绘制一些东西，比如按钮或标签。所以视图就是你可以看到的东西，而视图控制器则是垂帘听政的幕后操控者。
很多初学者将视图控制器对象命名为类似FirstView或者MainView。千万别这么做，这样会把你自己搞糊涂的！如果一个对象属于视图控制器，应该将其命名为"ViewController'，而不是"View"。

有时候我在想，苹果干毛要在view Controller的前面加个view，直接叫controller不是更省事吗。因为视图控制器不仅仅会控制视图，还会控制你的数据模型，它是二者之间的桥梁。
不过想归想，既然苹果这么命名了，我们就得跟着做，除非你把它买下来。

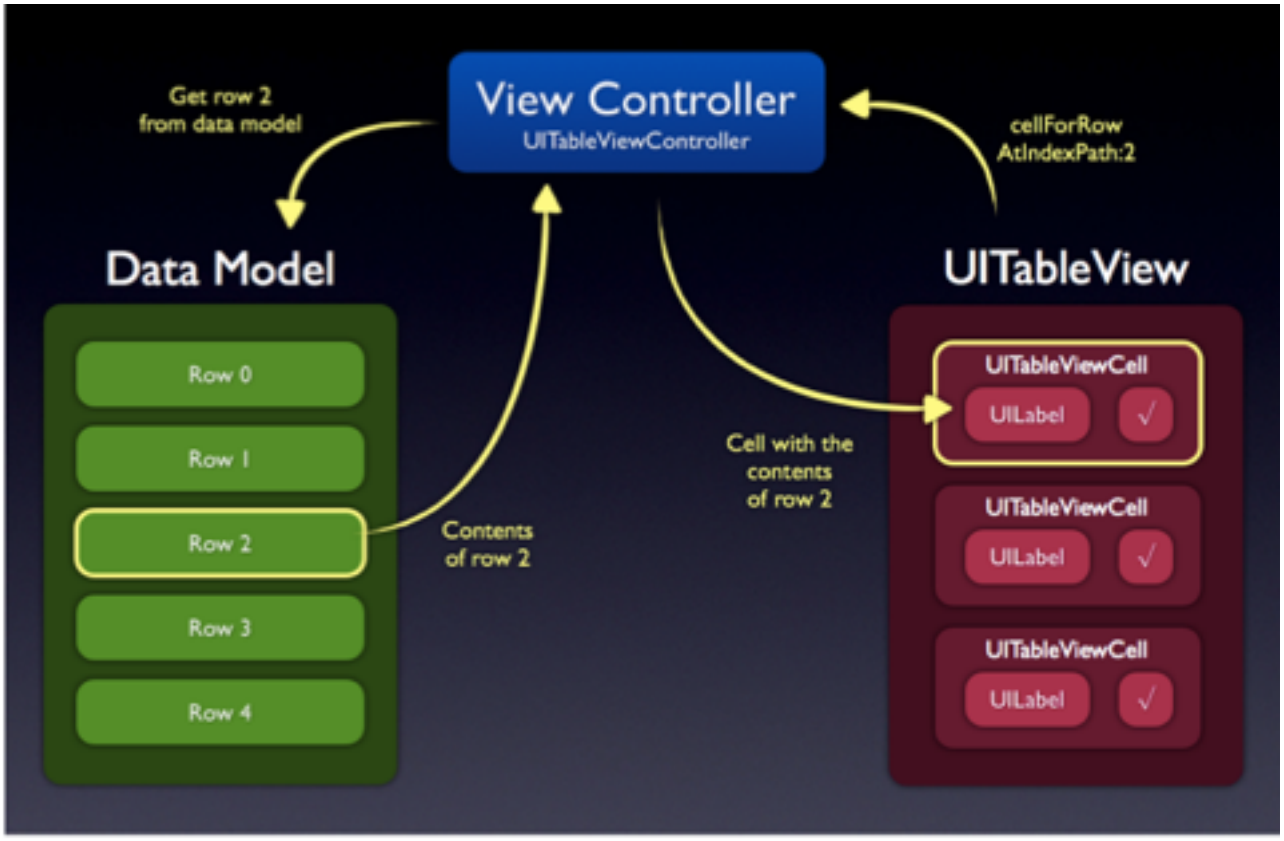
关于MVC的更多知识，可以看看维基和百度百科：
http://en.wikipedia.org/wiki/Model_view_controller
http://baike.baidu.com/view/5432454.htm?fromId=31&redirected=seachword

好了，理论知识学习时间够多了，还是来进行一些实际操作吧。

创建数据模型

到目前为止我们已经成功的把一堆伪数据塞到表视图里面去了。正如上一课提到的，我们不能使用cell来记住数据的状态，因为cell随时可能会被重用，其中老的内容会被新的内容所替代。cell是视图的一部分，它的作用是用来显示数据，但真正的数据来自另一个地方：数据模型。

再次强调，row（行）属于数据，而cell（表格）属于视图。表视图控制器通过实现表视图的数据源和代理方法将二者关联在一起。



通过上面这个图我们可以看到，表视图控制器（数据源）从数据模型中获取数据，然后将其填充到cell表格中显示出来。

在我们这个应用中，数据模型主要是代办事务的清单。每个代办事务都会在表视图中占据一行。对于每个代办事务对象，我们需要存储两类信息，分别是具体的文本内容和是否已被勾选。也就是说对每一行数据我们都需要存储两类信息，因此需要为每一行创建两个变量。

为了简便起见，我先演示一种偷懒的做法。请千万注意：这种方法是不可取的。它的确可以完成任务，但实现方法非常愚蠢。尽管这种方法不靠谱，我还是建议你跟着做，至少把代码直接拷贝到Xcode里面，然后运行应用。你需要先了解为神马这种方法是不好的，然后再来学习真正有效的方法。

终于到了代码时间了~

打开Xcode，切换到ChecklistsViewController.m，然后在@implementation这一部分代码中添加以下内容：

```
@implementation ChecklistsViewController{

    NSString *_row0text;
    NSString *_row1text;
    NSString *_row2text;
    NSString *_row3text;
    NSString *_row4text;
}
```

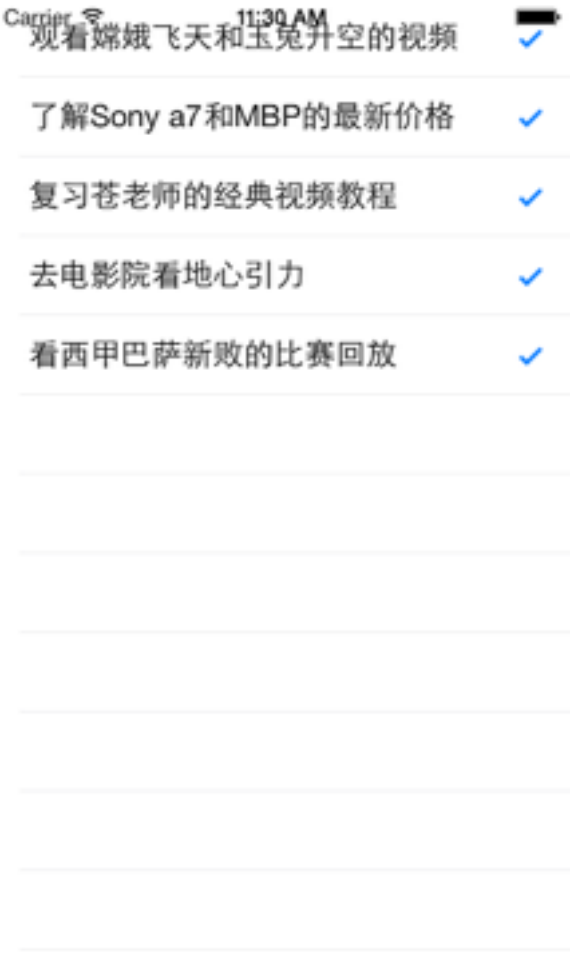
这里我们定义了五个字符串变量，看变量名称你应该知道是神马意思。
接下来更改viewDidLoad方法中的代码如下：

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

    _row0text = @"观看嫦娥飞天和玉兔升空的视频";
    _row1text = @"了解Sony a7和MBP的最新价格";
    _row2text = @"复习苍老师的经典视频教程";
    _row3text = @"去电影院看地心引力";
    _row4text = @"看西甲巴萨新败的比赛回放";

}
```

然后更改之前的两个数据源方法的代码如下：

```
-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{
    return 5;
}

-(UITableViewCell*)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath{

    UITableViewCell *cell =[tableView dequeueReusableCellWithIdentifier:@"ChecklistItem"];
    UILabel *label = (UILabel*)[cell viewWithTag:1000];

    if(indexPath.row  == 0){
        label.text = _row0text;
    }else if(indexPath.row  == 1){
        label.text = _row1text;
    }else if(indexPath.row  == 2){
        label.text = _row2text;
    }else if(indexPath.row  == 3){
        label.text = _row3text;
    }else if(indexPath.row  == 4){
        label.text = _row4text;
    }

    return cell;
}
```

好了，编译运行应用看看。



上面还是显示了5行内容，似乎没什么大不了的。
那么我们刚才究竟做了神马？对每一行数据，我们都定义了一个实例变量用于存储该行的文本内容。这5个实例变量就是我们的数据模型。
是的，Don't panic! 数据模型没什么大不了的，我们已经创建了一个最简单的数据模型。

在cellForRowAtIndex方法中，我们根据不同的indexPath.row属性来设置文本标签的不同内容。
好了，接下来我们可以修补一下上一课遗留的勾选逻辑问题了。
我们不再希望在cell上面切换勾选标志，而是在row行上面切换。因此，我们需要添加5个新的实例变量来记住每一行数据的勾选状态。

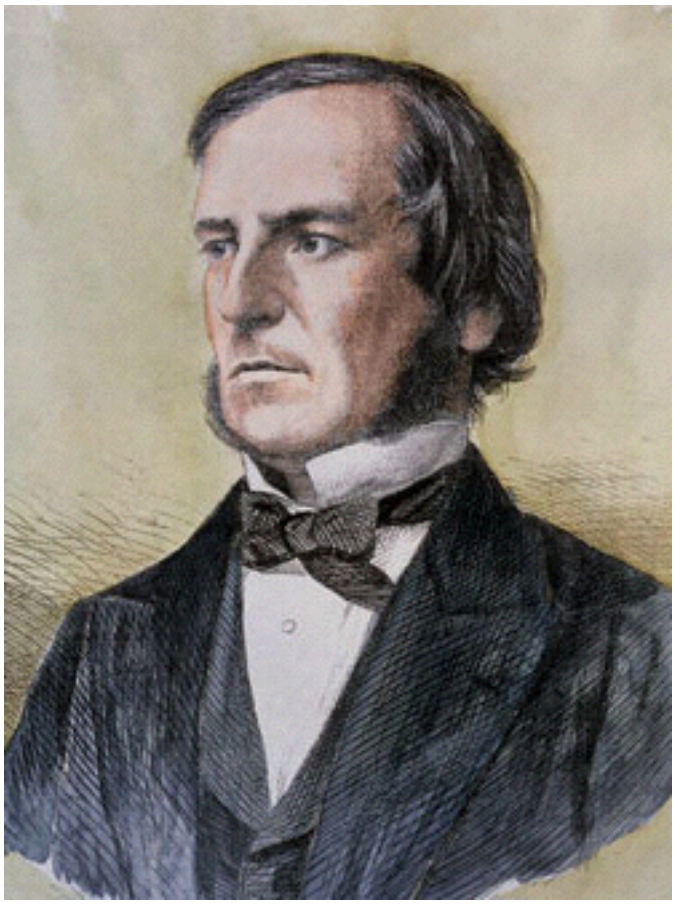
在@implementation部分添加5个实例变量：

```
@implementation ChecklistsViewController{

    NSString *_row0text;
    NSString *_row1text;
    NSString *_row2text;
    NSString *_row3text;
    NSString *_row4text;


    BOOL _row0checked;
    BOOL _row1checked;
```

```
    BOOL _row2checked;
    BOOL _row3checked;
    BOOL _row4checked;
}
```

这里又出来了一个新东西，BOOL类型变量。BOOL也是一种数据类型，就像int和NSSTring一样，只不过它的可能数值只有两个：YES和NO。在其它编程语言中，可能的数值会是true和false，比如C++和C，但是Objective-C则使用简单的YES和NO(注意全部要大写）。

BOOL是boolean的缩写，是为了纪念伟大的英国绅士乔治.布尔（George Boole）。他时19世纪的最重要的数学家之一。1854年，他出版了著名的The Laws of Thought，其中介绍了现在以他的名字命名的布尔代数。如果你曾经受过计算机专业的荼毒，或许在一门名为《离散数学》的课程中感受过。
这个绅士是下面这个样子的。



关于他的更多信息，请参看维基百科：
http://en.wikipedia.org/wiki/George_Boole

布尔是个伟大的NB人物，他所发明的布尔代数构成了当年计算机科学的基础，现在的计算机使用0和1也要归功于他。
好了，总而言之，我们会使用BOOL变量来保存某个东西是对（YES)或者不对（NO）的信息。在iOS开发中，布尔类型的变量通常用is或者has开头，比如hasIceCream。

在上面的例子中，当某一行数据有自己的勾选标志时，实例变量_row0checked的值是YES，反之是NO。其它几个变量类似。

现在我们需要更改用于处理用户触碰行数据的代理方法了。
更改didSelectRowAtIndexPath的代码如下：

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

    UITableViewCell *cell =[tableView cellForRowAtIndexPath:indexPath];

    BOOL isChecked = NO;

    if(indexPath.row ==0){

        isChecked = _row0checked;
        _row0checked = !_row0checked;

    }else if(indexPath.row ==1){

        isChecked = _row1checked;
        _row1checked = !_row1checked;

    }if(indexPath.row ==2){
        isChecked = _row2checked;
        _row2checked = !_row2checked;


    }if(indexPath.row ==3){

        isChecked = _row3checked;
        _row3checked = !_row3checked;

    }if(indexPath.row ==4){


        isChecked = _row4checked;
        _row4checked = !_row4checked;
    }

    if(isChecked){
        cell.accessoryType = UITableViewCellAccessoryNone;
    }else{
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    }

    [tableView deselectRowAtIndexPath:indexPath animated:YES];
}
```

在上面的代码中，我们首先根据indexPath.row的属性值来找到用户所触碰的行，然后找到对应的用于判断行是否被选中的实例变量。第一行就是_row0checked,第二行是_row1checked，其它以此类推。我们把它的值保存在临时变量isChecked中。然后用底部的方法根据isChecked的值来判断某个cell上是否需要设置或者移除勾选标志。

通过下面的这行代码：

```
_row0checked = !_row0checked;
```

我们让实例变量的值反转过来。
好吧，这里又出现了一个新的操作符!，或者逻辑非操作。对于BOOL类型的变量还有其它几种逻辑操作符，比如and（与）和or（或），很快我们就会碰到。
!这个操作符的作用是，让变量的值反转过来。比如本来_row0checked的值是YES,使用!可以让它的值变为NO。或者反过来!NO就是YES。
可以把!看做not（不是）的意思，不是yes就是no,不是no就是yes，我们在绕口令吗?

再次编译运行应用，然后看看效果。。。好吧，出问题了。我们得在一行上触碰好几次才会让勾选标志消失，还不如以前呢。

bug 去哪儿了？原因很简单：对于BOOL类型的变量，如果我们不设置初始值，它的默认值是NO。所有_row0checked和其它变量的初始值是NO，也就是没有勾选标志，但表视图中的prototype cell上已经有一个勾选标志了。
也就是说：数据模型（row checked变量）和视图（cell里面的勾选标志）不匹配。

肿么办?

可以有几种方法来解决这个问题，我们可以设置BOOL类型变量的初始值为YES，或者从prototype cell上去掉勾选标志。但这两种方法都不是很保险，因为问题的关键不是我们是否初始化了变量，或者是prototype cell的设计有误，而是我们忘了在cellForRowAtIndexPath中正确的设置勾选标志。

当我们请求一个新的cell时，要记住需要配置它所有的相关属性。对dequeueReusableCellWithIdentifier方法的iaoyong会返回一个cell,也就是带有勾选标志的行曾用过的cell，因此如果新的行没有勾选标志，我们需要先从cell中删除它。

可能你还稍有点疑惑，不过没关系，先跟着来操作，随着对Table View表视图的更深入认识，这些疑惑都会烟消云散的。

还是回到Xcode，在ChecklistsViewController.m中，在cellForRowAtIndexPath方法之前添加一个新的方法如下：

```
-(void)configureCheckmarkForCell:(UITableViewCell *)cell atIndexPath:(NSIndexPath *)indexPath{

    BOOL isChecked = NO;
    if(indexPath.row ==0){

        isChecked = _row0checked;

    }else if(indexPath.row ==1){
```

```
        isChecked = _row1checked;


    }if(indexPath.row ==2){
        isChecked = _row2checked;


    }if(indexPath.row ==3){

        isChecked = _row3checked;

    }if(indexPath.row ==4){

        isChecked = _row4checked;
    }
    if(isChecked){
        cell.accessoryType = UITableViewCellAccessoryNone;
    }else{
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    }

}
```

在这个新方法中，我们会根据indexPath的属性值来查找某个特定行对应的cell，然后根据row checked变量的值来设置勾选标志是否可见。

接下来我们要在cellForRowAtIndexPath方法中调用这个方法。

更改cellForRowAtIndexPath方法的代码如下：

```
-(UITableViewCell*)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath{

    UITableViewCell *cell =[tableView dequeueReusableCellWithIdentifier:@"ChecklistItem"];
    UILabel *label = (UILabel*)[cell viewWithTag:1000];

    if(indexPath.row  == 0){
        label.text = _row0text;
    }else if(indexPath.row  == 1){
        label.text = _row1text;
    }else if(indexPath.row  == 2){
        label.text = _row2text;
    }else if(indexPath.row  == 3){
        label.text = _row3text;
    }else if(indexPath.row  == 4){
        label.text = _row4text;
    }
```

```
    [self configureCheckmarkForCell:cell atIndexPath:indexPath];

    return cell;
}
```

再次编译运行，现在应用算是可以正常工作了。初始状态下所有的行都处于非选中状态。当用户触碰某一行时会将其选中，再次触碰就会取消勾选。现在row和cell已经完全同步了。

好吧，为神马这里要把configureCheckmarkForCell弄成一个独立的方法？这里因为我们也可以使用它来简化didSelectRowAtIndexPath方法。

使用以下代码替换tableView:didSelectRowAtIndextPath:方法：

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

    UITableViewCell *cell =[tableView cellForRowAtIndexPath:indexPath];


    if(indexPath.row ==0){

        _row0checked = !_row0checked;

    }else if(indexPath.row ==1){

        _row1checked = !_row1checked;

    }if(indexPath.row ==2){
        _row2checked = !_row2checked;


    }if(indexPath.row ==3){
        _row3checked = !_row3checked;

    }if(indexPath.row ==4){
        _row4checked = !_row4checked;
    }

    [self configureCheckmarkForCell:cell atIndexPath:indexPath];

    [tableView deselectRowAtIndexPath:indexPath animated:YES];
}
```

OK,现在我们不再直接设置cell的勾选标志，而是通过切换数据模型的checked状态，然后调用configureCheckmarkForCell方法来更新视图对象。

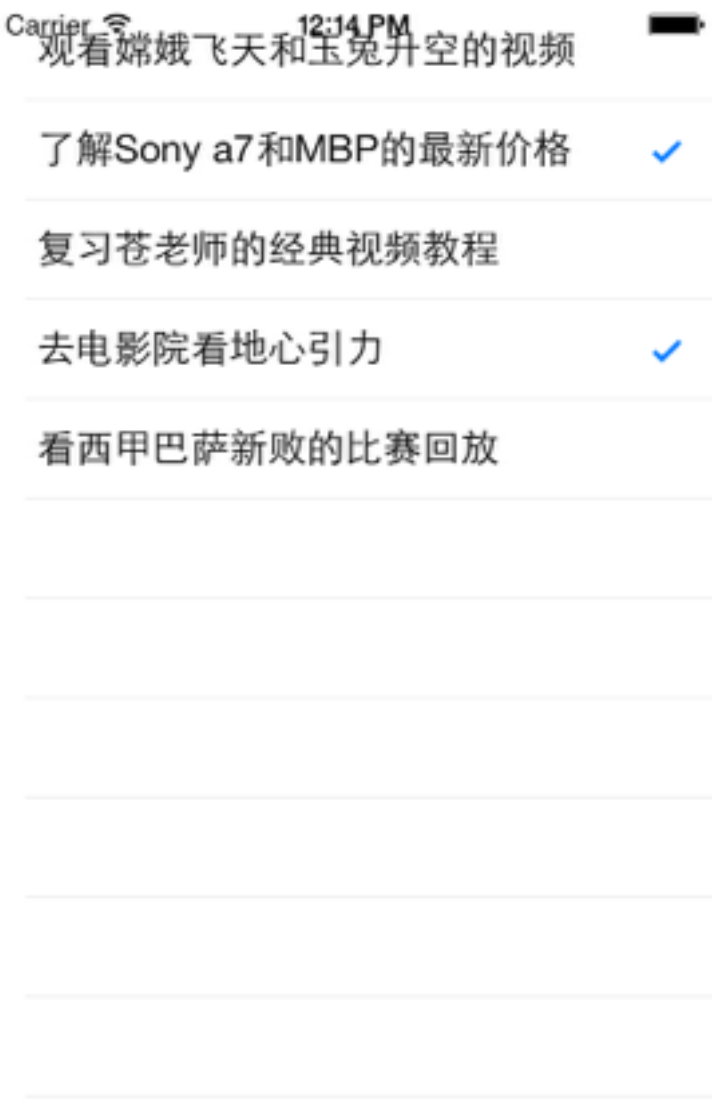
再次编译运行，一切都OK了。

好了，为了确保之前所做的一切都OK，我们再尝试更改下viewDidLoad方法的代码如下：

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

    _row0text = @"观看嫦娥飞天和玉兔升空的视频";
    _row1text = @"了解Sony a7和MBP的最新价格";
    _row2text = @"复习苍老师的经典视频教程";
    _row3text = @"去电影院看地心引力";
    _row4text = @"看西甲巴萨新败的比赛回放";


    _row1checked = YES;
    _row3checked = YES;
}
```

这里唯一的改动就是让第一行和第三行在初始状态下被选中。

试着运行下应用感受下。Oh Yeah!



好了，这种方法看来是有效的。在这一篇的内容中，我们复习了MVC 的概念，然后尝试着创建了一个最简单的数据模型。好吧，这里只有5行数据比较好办，如果有100行呢？还用这种蠢办法吗？

显然不会，该怎么做呢？预知后事如何，且听下回分解。

还是先送上福利吧。

天冷了，女神们可以考虑这件防寒装，秒杀所有渣男的目光

话说最近哥所在的地方连续重污染，该肿么办？这是必须得上空气净化器的节奏啊。

看来代办事务里面又要加一条，每天准时查看PM指数。