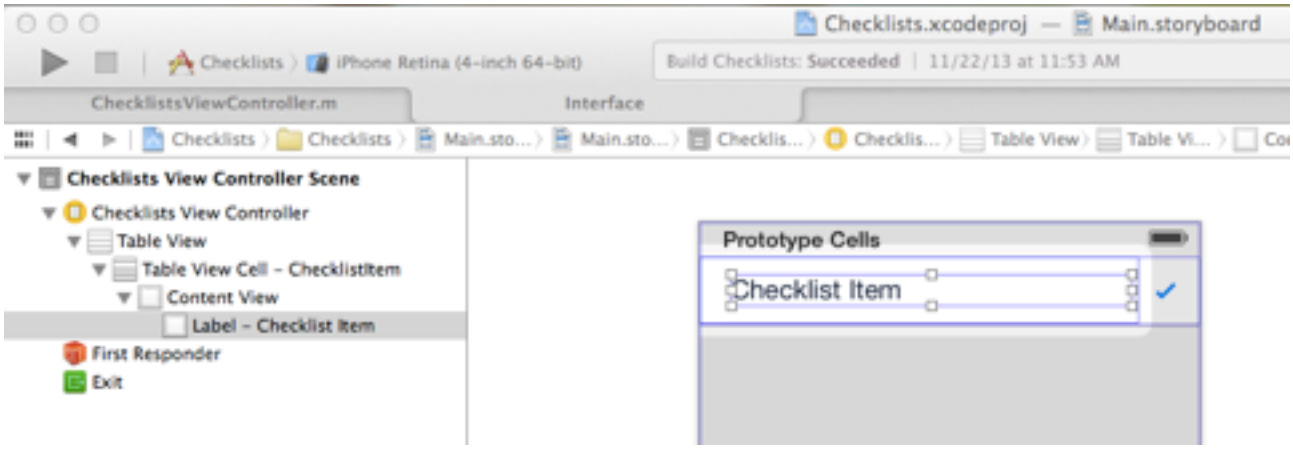
从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter2

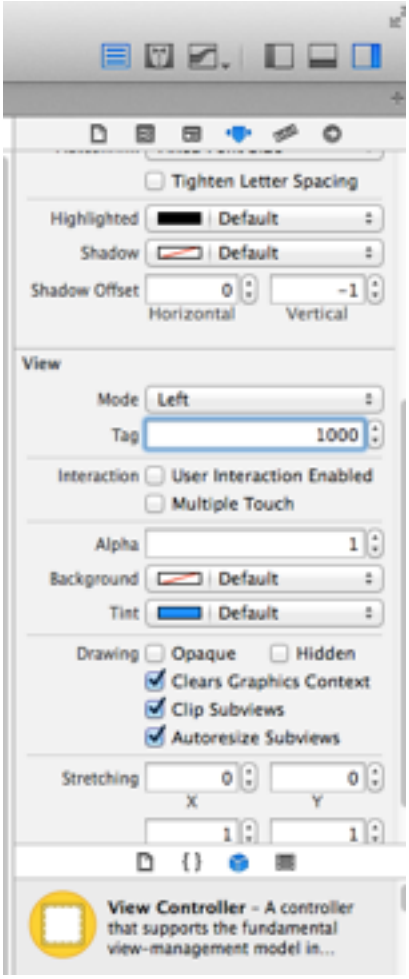
欢迎继续我们的学习。
在上一章的学习中，我们成功的在界面中添加了一个表视图，然后让它显示了几行数据信息，虽然看起来是完全相同的。

接下来让我们做点调整，起码先让每一行显示不同的内容吧。

在Xcode中打开storyboard文件，然后选择table view cell中的标签label。如果你怕选择错了，可以在左侧的视图元素列表中点击选择。



然后在Xcode右侧的面板中切换到Attributes Inspector，将Tag这一栏的数值设定为1000.



设置这样一个tag（标记）究竟有神马用处呢？通过tag标记我们可以给用户界面中的视觉元素一个数字标识符，这样在后续需要使用到它的时候可以很容易找到。那么这里为神马要设置为1000呢？没神马特别道理，只要这个数字不是0就行（因为0是标记的默认数值）。如果你愿意，用你最习惯的1024也可以，你懂的。

注意：一定要确定你的标记值是设置在标签上。初学者最长久的错误就是给table view cell设置了标记值，而不是标签。这样最后的结果就不是你所期待的了。

想上头条可以理解，不过如果你把自己标记成了"汪峰"，即便后面调用了正确的方法，比如推新歌，苦逼爬山。。。不管怎样，很遗憾今生你依然与头条无缘，哪怕顶上月亮也没用。

接下来在Xcode中打开ChecklistViewController.m，更改cellForRowAtIndexPath方法如下：

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

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

    if(indexPath.row ==0){
        label.text = @"观看嫦娥飞天和玉兔升空的视频";
    }else if(indexPath.row ==1){
        label.text = @"了解Sony a7和MBP的最新价格";
    }else if(indexPath.row ==2){
        label.text = @"复习苍老师的经典视频教程";
    }else if(indexPath.row ==3){
        label.text = @"去电影院看地心引力";
    }else if(indexPath.row ==4){
        label.text = @"看西甲巴萨新败的比赛回放";
    }

    return cell;
}
```

在这个方法体中，第一行代码我们之前已经了解过了，它的作用就是获取prototype cell的拷贝（不管是新的还是回收利用的），然后把它赋予新创建的一个cell本地变量，其类型是UITableViewCell。
接下来的就是新东西了，首先是这一行：

```
    UILabel *label = (UILabel*)[cell viewWithTag:1000];
```

这里我们请求获取cell中的标记为1000的子视图，这个标记就是我们刚才在storyboard中在标签上所设置的。通过这种方法，我们就获取了一个到该UILabel标签对象的引用。
实际上，通过使用tag标记的方式来获取到某个视觉元素的引用是非常方便的，可以省掉了声明@property属性变量的步骤。

小练习：

好吧，为神马这里我们不在视图控制器中添加一个IBOutlet属性声明，然后把cell的标签和storyboard中的outlet关联在一起呢？

答案是：在table视图中可能不止一个cell，而每个cell都会拥有自己的标签。如果我们使用之前的方法，将prototype cell中的标签和视图控制器中的某个outlet关联在一起，那么该属性只能指向其中一个cell的标签，而不是所有。
需要注意的是，label标签属于cell的子视图，而不是属于视图控制器，因此我们不能为它创建一个outlet。（当然，并不是说我们就不能对table view cell使用属性。在后续的教程中我们将学到，但在这里就暂时别管了）。

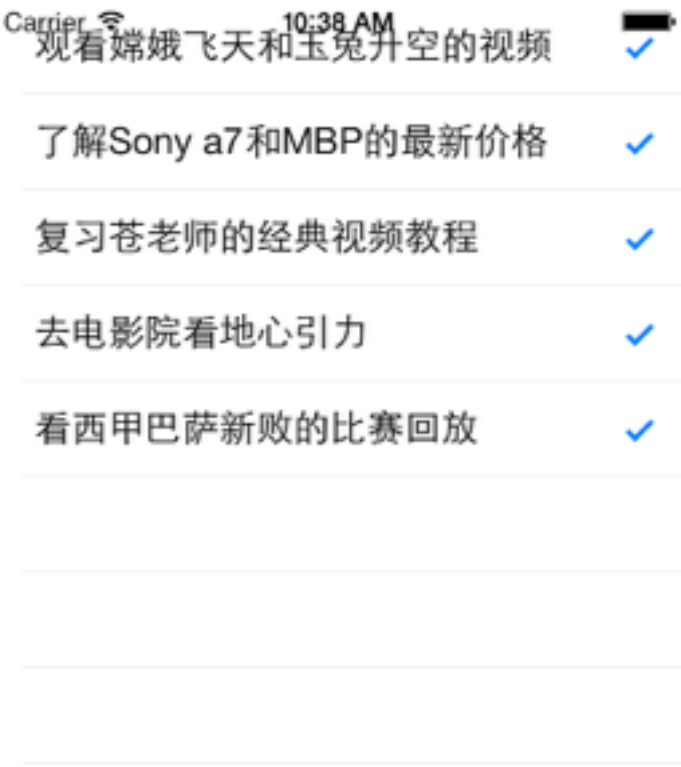
继续回到代码。

接下来的几行代码应该不会让你很头大。

```
if(indexPath.row ==0){
    label.text = @"观看嫦娥飞天和玉兔升空的视频";
}else if(indexPath.row ==1){
    label.text = @"了解Sony a7和MBP的最新价格";
}else if(indexPath.row ==2){
    label.text = @"复习苍老师的经典视频教程";
}else if(indexPath.row ==3){
    label.text = @"去电影院看地心引力";
}else if(indexPath.row ==4){
    label.text = @"看西甲巴萨新败的比赛回放";
}
```

之前我们已经学过if- else if -else的语法结构。这里程序检查了indexPath.row的属性值，其中包含了行编号，然后根据不同的行编号赋予标签中的文本不同的值。
注意，第一行的index属性值是0，而不是1。

好了，现在可以运行应用，看看我们有哪些事情要做的~



通过改写的cellForRowAtIndex方法，我们向表视图提供了所需的数据。首先获取了一个UITableViewCell对象，然后根据NSIndexPath的行编号属性更改cell中的内容。

如果你闲得无聊，可以试着往表视图中填充100行数据。

更改以下两个方法的内容为：

```
-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{
    return 100;
}

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

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

    if(indexPath.row %5 == 0){
        label.text = @"观看嫦娥飞天和玉兔升空的视频";
    }else if(indexPath.row %5 == 1){
        label.text = @"了解Sony a7和MBP的最新价格";
    }else if(indexPath.row %5 == 2){
        label.text = @"复习苍老师的经典视频教程";
    }else if(indexPath.row %5 == 3){
        label.text = @"去电影院看地心引力";
    }else if(indexPath.row %5 == 4){
        label.text = @"看西甲巴萨新败的比赛回放";
    }

    return cell;
}
```

以上方法的内容和之前没有太大区别，只是让numberOfRowsInSection返回100，同时在cellForRowAtIndexPath中使用了一个新的方式来判定该显示什么内容。

```
if (indexPath.row % 5 == 0) {
                    } else if (indexPath.row % 5 == 1) {
                    } else if (indexPath.row % 5 == 2) {
                    } else if (indexPath.row % 5 == 3) {
                    } else if (indexPath.row % 5 == 4) { }
```

这里用到了模操作符 %，如果你的小学数学还没忘光光的话，可能对它还有几分印象。模操作其实就是返回除法计算的余数。比如13 %4 =1，因为13除以4的结果是3.25，用整数来表示的话除法的结果是3，然后余数是1.而12%4=0，因为没有余数。

这样一来，第1行，第6行，第11行，第16行，等等都会显示"观看嫦娥飞天和玉兔升空的视频"，第2行，第7行，第12行，等等都会显示"了解Sony a7和MBP的最新价格"，以此类推。

想来你已经很清楚，每隔5行所显示的文本内容都会再次重复。这里我们会显示100行内容，如果你要搞上一大堆普通的if-else if -else也可以，但显然我们可以使用模操作来简化这个过程。

第一行： 0%5 = 0
第二行： 1%5 =1
第三行： 2%5 =2
第四行： 3%5 =3
第五行： 4%5 =4
第六行： 5%5 =0 （和第一行相同，从这里开始所显示的内容会开始循环）

第七行： 6%5 =1（和第二行相同）
第八行： 7%5 =2（和第三行相同）
第九行： 8%5 = 3（和第四行相同）
第十行： 9%5 = 4 （和第五航相同）
第十一行： 10%5 =0（和第一行相同，再次循环之前的内容）
第十二行： 11%5 =1 （和第二行相同）

。。。。。。
后面就不再折腾了，相信你百分百可以看懂了。

假如你还是看不懂，也没关系，只要知道这里我们用了一个小小的技巧快速填满了整个表。

运行应用，可以看到类似下面的界面：

思考：
为了显示这100行数据，我们用到了多少个table view cell呢？

答案是：

虽然我们有100行数据，但在屏幕上最多同时可以容纳14个cell。
等等，哥刚才仔细数了数，不是只有13个cell吗？没错，如果表中的内容静止，的确我们只看到13个cell，但如果你上下滚动表视图，会发现有时上面的cell还在显示，而新的cell已经在底部出现了。因此至少需要14个cell。

如果你滚动表视图的速度足够快，或许可能表视图需要更多的临时cell，这一点我不确定。不过这一点重要吗？不重要吗？是的，不重要。你只需要表视图会替你完成这个任务就行了。我们要做的事情是，当表视图需要一个cell的时候向它提供，同时用相应行的数据信息来填满这个cell就好了。

继续继续。

触碰表视图中的行

如果你尝试触碰某一行数据，会发现它的背景会变成灰色。

但是即便我们再次触碰，后面的勾选状态依然存在。接下来我们需要做点小小的调整，这样触碰某一行数据时会开启或关闭后面的勾选状态。

对行数据的触碰是由表视图的delegate（代理）来处理的。之前我们提到过，在iOS开发中，经常会让某些对象替代另一些对象来完成某些工作。表视图的数据源就是一个很好的例子，不过表视图还需要另外一个小帮手来处理其它的工作，也就是table view delegate。

关于代理模式

代理的概念在iOS开发中运用的非常广泛。其实不光是iOS(Objective-C)开发中会用到代理/委托，在C#,Java等开发语言中也会经常碰到它。不过想来大家也没怎么接触过其它的开发语言，所以这里肯定是要多废话几句的。

在现实生活中，我们经常接触到代理/委托的概念，比如我们会委托房产中介来出租/出售房子，如果你是x峰，还会找个经纪人或者公关公司来解决上头条的难题。又比如有些公司会把自己不擅长或者不想费心思的某些工作（比如行政、公关、财务）外包给其它公司来完成。

在代码世界中，某些对象经常需要借助另一个对象的帮助才能完成某些特定任务。通过这种术业有专攻的专业分工模式，可以让整个系统尽可能简单清晰明了。每个对象都专注在自己所擅长的工作中，然后让其它对象处理各自所擅长的事务。

对于表视图的使用是iOS开发中如何使用代理模式的绝佳示范。

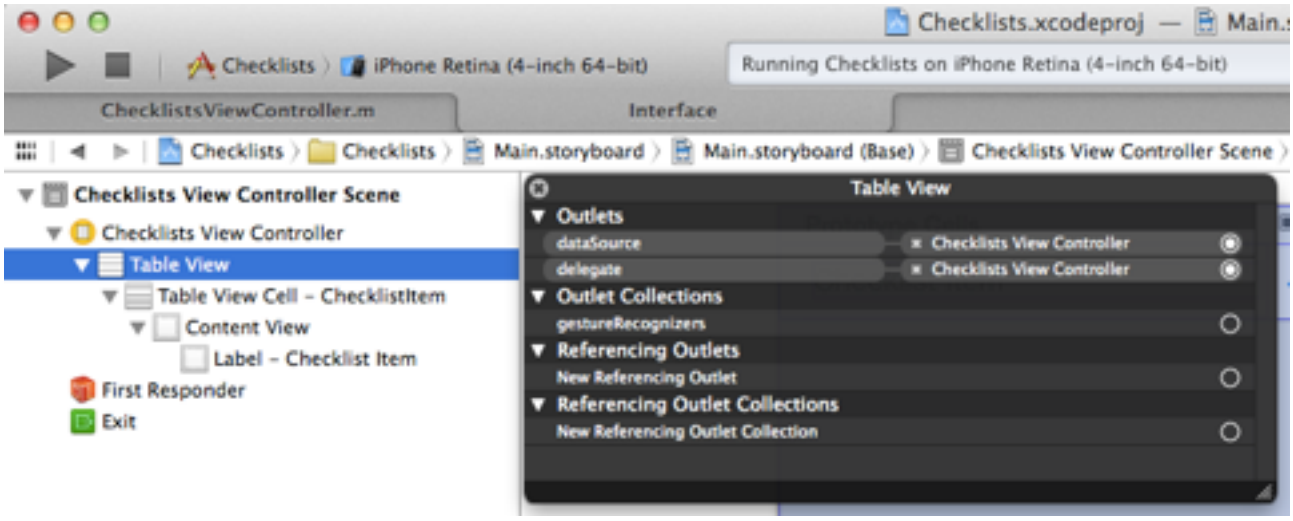
因为某个应用都会用到不同形式的数据，因此表视图需要具备处理不同类型数据的能力。但表视图只是一个视觉元素，我们没有必要把表视图的功能设计得庞大而臃肿，或者进行人为的定制调整以适应不同应用的需求。反之，苹果UIKit的设计者选择把向cell中填充数据的艰巨工作交给了另一个对象，也就是数据源。

表视图并不在乎谁会是它的数据源，或者你的应用中可能会处理到何种类型的数据。它只需要发送一个cellForRowAtIndexPath消息，然后返回一个cell即可。通过这种模式，可以让表视图本身变得简单，不过这样一来，具体处理数据的工作就交给更擅长的你来完成了：你需要在代码中实现。

与此类似，表视图知道如何识别用户何时触碰了某一行，但对此的响应却完全取决于不同应用的不同设计。在这个应用中，我们要做的是开启/关闭勾选标志，而另一个应用则可能处理完全不同的事务。通过代理机制，表视图只需要在用户触碰某一行时发送一个消息，然后代理对象将负责处理其它的工作。

通常情况下某个视觉元素只有一个代理对象，不过表视图将代理的工作分给两个不同的帮手：UITableViewDataSource负责将行数据填充到表视图中，而UITableViewDelegate则负责处理用户对行的触碰及其它工作。（其实两者的任务分配不是那么严格，为了了解各自的职责，最好的方法是查询Xcode内置的帮助文档）。

让我们回到Xcode,打开storyboard文件，然后按住Ctrl键，点击Table View，这时你回看到表视图的dataSource和delegate都和视图控制器关联在一起。这其实是UITableViewController的标准实践。（当然你可以选择在通用的视图控制器中使用表视图，但这样的话就需要你手动关联数据源和delegate代理）。



接下来切换到ChecklistsViewController.m，在@end前添加以下方法：

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

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

运行应用，然后触碰某一行，你会看到该行的背景快速变灰，然后再次恢复之前的状态。

接下来让我们修改didSelectRowAtIndexPath方法，从而在触碰某行的时候来开启/关闭选中标示：

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

    UITableViewCell *cell =[tableView cellForRowAtIndexPath:indexPath];

    if(cell.accessoryType == UITableViewCellAccessoryNone){


        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    }else{
        cell.accessoryType = UITableViewCellAccessoryNone;
    }


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

在以上的方法中，我们首先获取了用户所触碰的UITableViewCell对象。要做到这一点很简单，我们只需要问表视图：你给我的cell的indexPath是神马呢？请注意这里我们直接对table view调用cellForRowAtIndex方法，而不是对视图控制器。二者都有这个同名的方法，但其作用和本质是不同的。）

接下来我们会查看cell的附属属性，这里是accessoryType属性。如果当前是"none",就需要让附属类型更改为勾选标志，如果已经是勾选标志了，那么就取消勾选。

此时编译运行应用，就可以通过触碰某行的方式来开启/关闭对行数据的勾选了。

注意：

如果你发现只有当你选择了另一行的时候才会看到勾选标志，那么你需要检查一下代码，确保方法的名称是tableView:didSelectRowAtIndexPath:，而不是didDeselect!
之所以会出现这种低级错误，是因为Xcode的自动完成功能有时候会让你不小心输入错误的方法名称。

真不错，事务管理软件的功能原型就算是搞定了？且慢，假如你足够细心的话，很快就会发现一个小bug：

触碰某一行来取消勾选标志，然后滑动这一行到屏幕之外，然后再次滑动回来（快速，快，快！）你会发现勾选标志再次出现了！此外，其它行的勾选标志似乎同时消失了！这是神马情况？

好吧，这里又要谈到cell和row的区别了！
我们的确开启关闭了cell上的勾选标志，但是，这个cell有可能会被其它行重用。因此，是否被勾选应该是行的属性，而非cell的属性。使用cell的附属属性来记住勾选状态只是权宜之计，我们需要使用其它方法来记录每一行的选中状态。因此，接下来我们需要扩展数据源，让其使用恰当的数据模型。别害怕！Don't Panic！我们会慢慢来的。

好了，今天的学习就到此结束了。想想看，你已经成功完成了如此多的工作，是该给点福利了！

神马？你要炉石传说的激活码？！这个，真没有，哥也是花钱从某宝买的。。。
据说这个MM是炉石台服的配音，你是不是想弄个台服账号玩玩了？
“那就我上吧”，对，就这张牌，该你上了。