从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter5

欢迎继续我们的学习。

在经过了漫长的上一章学习后，今天的内容相对轻松，没有多少新东西，主要是一些脏活累活~

我们要对项目里面的代码做一些清洁工作，打开Xcode，切换到ChecklistsViewController.m,然后使用以下代码替换之前的方法：

```
-(void)configureCheckmarkForCell:(UITableViewCell *)cell withChecklistItem:(ChecklistItem
*)item{


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

}

-(void)configureTextForCell:(UITableViewCell *)cell withChecklistItem:(ChecklistItem *)item{


    UILabel *label = (UILabel *)[cell viewWithTag:1000];
    label.text = item.text;

}

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

    UITableViewCell *cell =[tableView dequeueReusableCellWithIdentifier:@"ChecklistItem"];


    ChecklistItem *item = _items[indexPath.row];

    [self configureTextForCell:cell withChecklistItem:item];
    [self configureCheckmarkForCell:cell withChecklistItem:item];

    return cell;
}

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

```
    UITableViewCell *cell =[tableView cellForRowAtIndexPath:indexPath];
    ChecklistItem *item = _items[indexPath.row];
    [item toggleChecked];


    [self configureCheckmarkForCell:cell withChecklistItem:item];

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

小练习：

仔细对比下修改后的代码和之前代码间的区别，看看哪些地方变了？为什么？

回答：

首先，configureCheckmarkForCell:atIndexPath:方法被改名为configureCheckmarkForCell:withChecklistItem:。好吧，如果你觉得这种方法名称看起来太可怕了，don't panic!
实际上，iOS SDK中很多对象和方法的名称都是又臭又长的，不过好处就是它们可以让你更容易理解各自的作用。幸运的是，Xcode提供了自动补充的特性，所以你只需要手动敲几个字符，就会自动向你提示剩下的部分，否则光敲代码就得让你头大了！

为什么要修改这个方法呢？
之前的方法接收一个indexpath参数，然后用它来找到所对应的ChecklistItem

```
ChecklistItem *item = _items[indexPath.row];
```

但无论在cellForRowAtIndexPath和didSelectRowAtIndexPath中，我们都已经做了同样的事情。为了避免重复工作，我们可以考虑直接将ChecklistItem对象作为一个参数来传递。

此外，我们还添加了一个configureTextForCell:withChecklistItem:方法，该方法的作用是设置cell中标签的文本内容。之前我们在cellForRowAtIndexPath方法中曾经完成过这项工作，不过把它单独放到一个方法里面显得代码结构更加清晰。

最后，didSelectRowAtIndexPath方法不再直接修改ChecklistItem的选中属性，而是通过调用一个名为toggleChecked的方法来完成该工作。
所以显然我们需要添加一个新的方法。

在Xcode中切换到ChecklistItem.h（注意不是ChecklistsViewController.h），然后在@end前添加方法的声明：

```
-(void)toggleChecked;
```

然后切换到ChecklistItem.m，在@end前添加该方法的实现代码：

```
-(void)toggleChecked{

    self.checked = !self.checked;
```

}

如你所见，该方法的作用其实之前在didSelectRowAtIndexPath中曾实现过，只不过这次我们把它添加为ChecklistItem对象的一个单独方法。好的面向对象设计法则是，尽量让对象管理自己的状态。比如在这里，之前是由视图控制器来完成状态切换工作，而这里改为由ChecklistItem自己来切换。

编译运行应用，如果一切正常的话，那么效果和之前应该是一样的~

好吧，你让哥改了这么多行代码，最后的结果竟然是一样的！

其实还是不太一样的。首先，整个代码结构更加清晰明了，可以有效避免bug的出现。通过使用数组，我们让程序变得更加灵活，现在表视图可以处理任意行的数据了。

在开发项目的过程中，我们经常会不断重新组织代码，术语党给了一个可怕的名词叫代码重构。因为在最开始写的时候，不可能100%都是完美的。然后程序猿开始不断添砖加瓦，直到整个代码结构

开始变得混乱，这时候就要修修补补做做清洁工作了。然后过了一段时间再次变得一团乱麻，然后苦逼的你又要开始清理一下。整个过程看起来永无休止（要不要这么苦逼？）

还有一些程序猿宣传自己从未清理过自己的代码。虽然有少数牛人可以做到从项目开始到最后始终保持代码结构的完美，但大多数开发者这样做的后果是让程序一团糟，对于后续维护的人来说简直就是一场灾难。

除非你的项目完成后这辈子都不会有机会碰到了（包括别人），否则还是别这么缺德。做人，还是厚道点好。

今日福利，小清新MM一张