从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter4

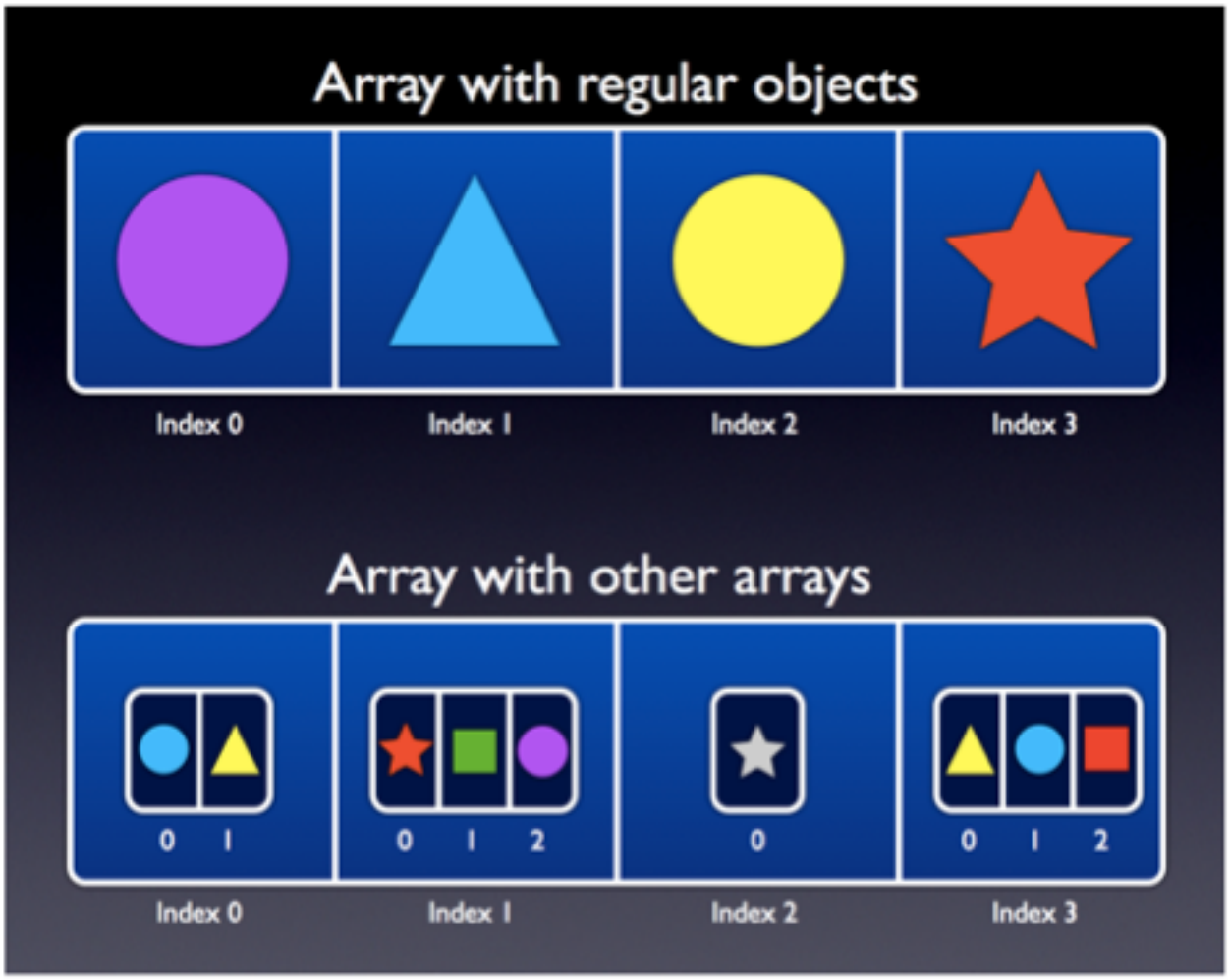
欢迎继续我们的学习。

在上一篇的内容中我们成功的创建了第一个数据模型，虽然是用最蠢的办法来实现的，但好歹也是实现了。
但之所以说这种方法很蠢，是因为我们这里只显示了5行数据，如果有100行甚至1000行数据要显示呢？难不成你要分别创建1000个变量来存储这些数据信息？
要知道程序猿通常是最会偷懒的一个群体，虽然他们经常和代码打交道，但其实是最懒得写代码的。

为了让自己可以更好的偷懒，于是一个新的事物诞生了，那就是-数组(array)!

关于数组(array)

不知道哪个喜欢卖弄悬殊的家伙把array翻译成了数组，千万不要顾名思义，array（数组）可不是数字的组合，确切的说是"对象阵列"，也就是一堆对象的有序排列。
我们现在已经知道了变量可以用来保存一个数值或者一个对象，而数组则可以保存多个数值或对象。当然，在Objective-C中，array本身也是一个对象（也就是NSArray对象），我们可以在这个对象里面放入变量。还有一件更酷的事情，因为array本身也是对象，所以我们可以在array数组中保存其它数组。



比如在上图中，上面的那个数组里面保存的是常规的对象，而下面的数组里面保存的则是其它数组。

所以从这个角度来看，array翻译成数组实在是太不知趣了。
好了，吐槽这些术语翻译也没用，谁叫你不是第一批接触国外计算机科学的人呢？说不定有人还会找出一大堆理由说这种翻译很有道理很有深度呢。但新手小白肯定不会这么认为。

这也是我为什么一再强调学习编程最好直接看英文书籍、文档和博客，因为老外很少装B，哪怕是成名已久的科学家在写科普类文章的时候仍然是时刻把小白的需求放在心上。

数组中的元素是使用数字来编号的，按照代码世界的惯例，当然还是以0开始。为了获取某个数组中的第一个对象，可以用类似[array objectAtIndex:0]的方法，或者更简单的方法是array[0]。需要注意的是，数组是"有序"排列的对象组合，因此对象在数组中的排列顺序是很重要的。如果我们要获取一个index编号为1的对象，就不能用array[0]。

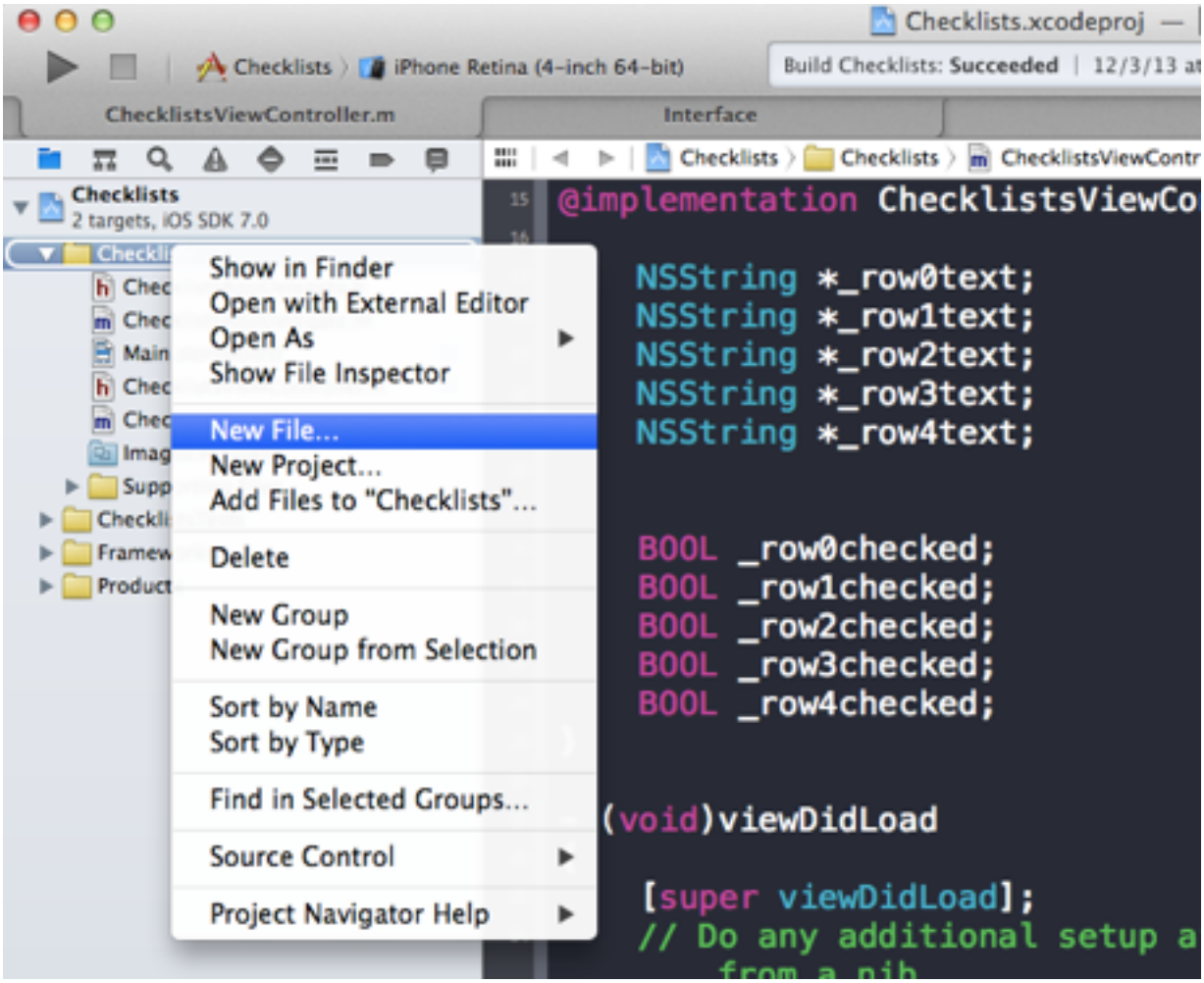
实际上在Objective-C中，NSArray还是一个collection(集合）对象。除了NSArray, NSDictionary和NSSet也属于集合类对象，它们的区别在于对象在集合中的排列组合方式不同。
以NSDictionary对象为例，里面所保存的对象都是以键值对的形式存在。好吧，一不小心我也装B了。
什么是键值对？如果你用过现实生活中的字典就会知道，字典中的内容形式都是词条和对词条的解释。也就是一个词条对应一个解释。键值对也是这个意思，一个主键对应一个数值。你可以把NSDictionay看做一个对象词典。只是里面的词条和解释都是对象。

NSSet相对好理解一些，也就是对象的集合。和NSArray不同的是，里面所保存对象的排列顺序不重要。打个不恰当的比方，NSArray就像超市的某一排酒架，里面放着的酒都是有序排列的。要拿其中的一瓶酒，只需要记住放在第几格就行了。但NSSet就没这么方便了，它就像一个大酒桶，我们把N瓶酒直接扔进去，那么取出来的究竟是哪一瓶酒就全凭运气了。

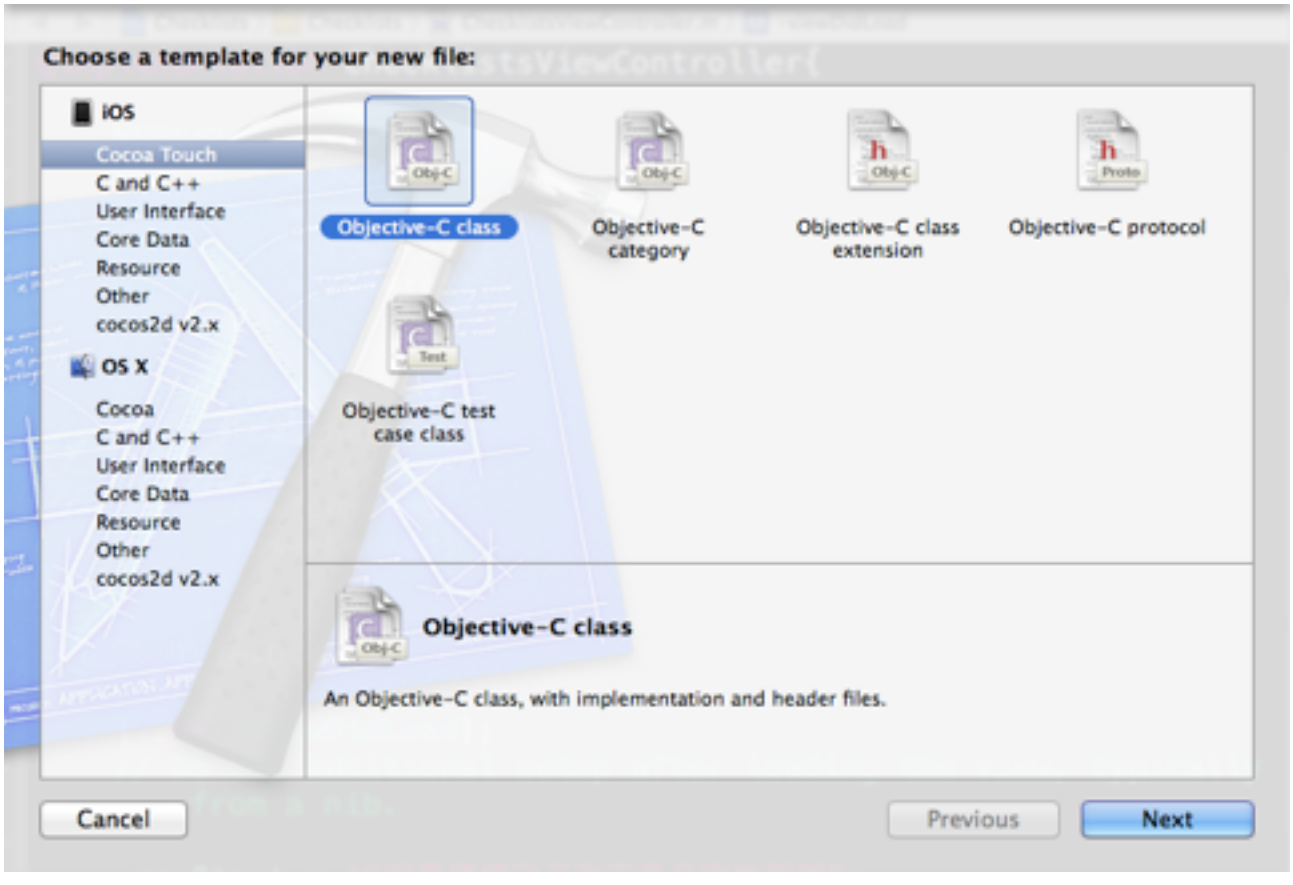
数组中对象的组织形式和表视图中的行有些类似，它们都是对象的有序排列，因此这里我们可以考虑把表视图中行数据的数据模型放到一个数组中。

数组可以存储对象，但现在行中包含两部分数据，文本和选中状态。如果我们可以让每一行只有一个对象就好办了，这样一来表视图中的行编号就直接等同于数组中的index编号。为了实现这一点，我们需要把文本内容和勾选状态整合到一个自创的新对象里面。

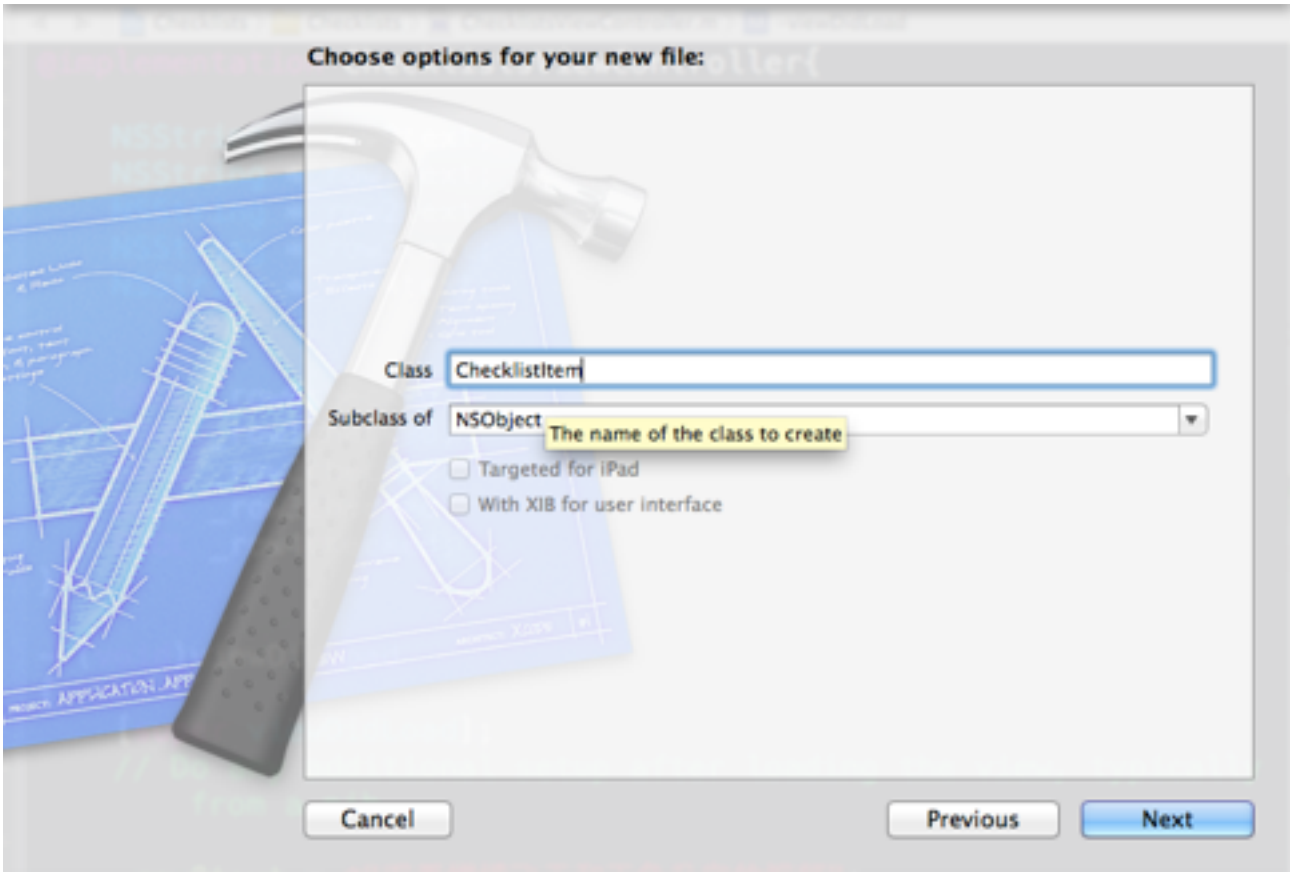
打开Xcode，在项目导航部分选中Checklists群组，然后右键单击。从弹出菜单中选择New File…



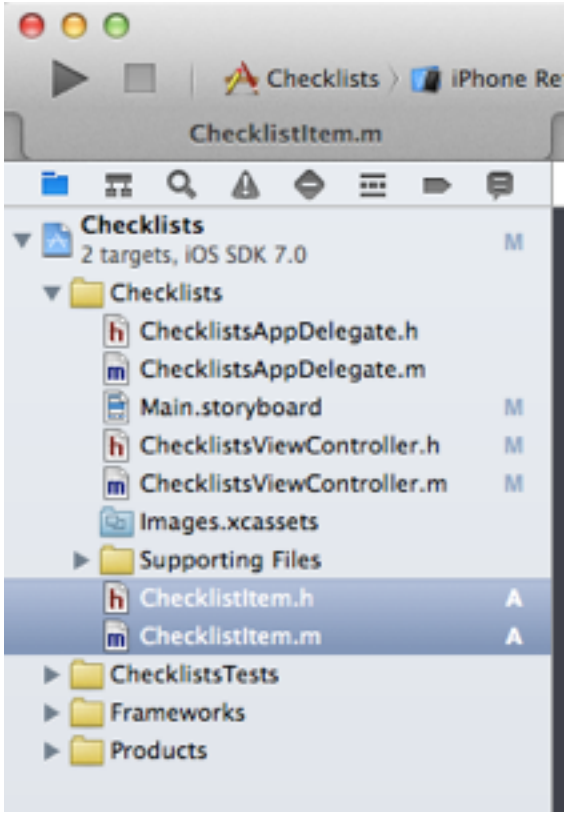
然后在Cocoa Touch部分选择Objective-C class:



接下来填充一些信息，
Class: ChecklistItem
Subclass of: NSObject



点击Next,然后Create就可以了。
现在我们已经创建了新的Checklist类（ChecklistItem.h和ChecklistItem.m）



切换到ChecklistItem.h，里面的代码如下：

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

@interface ChecklistItem : NSObject

@end
```

记住忽略上面那些绿色注释的语句，你可以在注释里面写YY小说，写情书甚至是入D申请书，但是不要浪费太多时间。

然后切换到ChecklistItem.m，其中的代码如下：

```
#import "ChecklistItem.h"

@implementation ChecklistItem

@end
```

好了，这其实就是在iOS中创建一个新对象的最简单方法。

接下来我们要将其完善。
首先切换到ChecklistItem.h，在@end这行代码前添加：

```
@property(nonatomic,copy)NSString *text;
@property(nonatomic,assign)BOOL checked;
```

这里我们添加了对象的两个属性，其中text属性用来保存代办事务项目的具体描述，而checked属性则用来判断是否需要显示一个勾选标志。

问题来了，为神马这里我们要把这两个数据项添加为属性变量，而不是实例变量？
原因是：
实例变量的生存空间局限于对象的内部，他们无法跨越时空被其它对象访问和使用。

在我们当前这个应用中，文本和勾选状态属性必须是其它对象可以访问的，这里的其它对象其实指的就是视图控制器。这些数据项是ChecklistItem对象的"public interface"（公用接口）的一部分，正因为此我们在@interface部分来添加这些属性变量的声明。

当然，和在前一系列教程（让不懂编程的人爱上iPhone开发）中不同的是，这两个属性变量没有IBOutlet在前面，是因为它们不属于outlets。只有我们需要将所定义的属性变量和Interface Builder中的视觉元素关联在一起的时候才需要在前面添加IBOutlet，从而让Xcode知道。
在这里两个属性变量和界面视觉元素无关，只是数据模型的一部分，因此不需要这么做。

好了，这样就行了。我们暂时还不需要对ChecklistItem.m做任何修改。ChecklistItem对象当前的主要作用就是将文本和勾选标志合并到一个对象之中。

如果你对这种组合方式觉得有点理解不能，可以考虑下自己的投资组合。假定你是个土豪，那么很有可能你的资产不会全部是土豪金，也不会全部是房产，更不会全部是现钞。。。很有可能你的资产中既有土豪金，还有房产，还有现钞，还有股票，还有期权，还有公司股份。假定你的资产是一个对象MyAsset,那么它可以有多个属性，

```
比如@property(nonatomic,assign)int amountOfGoldInOunce;
```

用盎司来计算的土豪金储备算一个。
又比如@property(nonatomic,assign) int valueOfRealEstateInRMB;
用RMB来计算的房产估价，当然，这个是波动的，和土豪金一样。
以此类推，这是这种对象组合似乎看起来比ChecklistItem更有实际意义一些~

在开始使用数组之前，先让我们使用ChecklistItem对象来替代视图控制器中的NSString和BOOL实例变量吧。

首先我们需要通知视图控制器需要用到ChecklistItem对象，否则它就会崩溃的。
这里只需要添加一行#import声明即可。

在Xcode中切换到ChecklistsViewController.m，然后添加以下代码：

#import “ChecklistItem.h"

这行代码的意思是，现在我们需要用到ChecklistItem这个对象，如果你想了解这个对象的任何信息，可以到ChecklistItem.h中查询。

接下来删除@implementation后面的NSString和BOOL实例变量声明，然后用ChecklistItem对象来替代：
@implementation ChecklistsViewController{

```
    ChecklistItem *_row0item;
    ChecklistItem *_row1item;
    ChecklistItem *_row2item;
    ChecklistItem *_row3item;
    ChecklistItem *_row4item;


}
```

好了，现在Xcode会提示你有大量的错误。Don’t panic！一切尽在掌握之中！
这是因为在视图控制器的一些方法中会用到之前所定义的实例变量。现在我们一股脑儿全部删除了，Xcode不疯掉才怪。

为了让Xcode不再冷眼相对，首先我们要修复这些错误。

更改viewDidLoad方法的代码如下：

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

    _row0item = [[ChecklistItem alloc]init];
    _row0item.text =@"观看嫦娥飞天和玉兔升空的视频";
    _row0item.checked = NO;

    _row1item = [[ChecklistItem alloc]init];
```

```
    _row1item.text =@"了解Sony a7和MBP的最新价格";
    _row1item.checked = NO;

    _row2item = [[ChecklistItem alloc]init];
    _row2item.text =@"复习苍老师的经典视频教程";
    _row2item.checked = NO;

    _row3item = [[ChecklistItem alloc]init];
    _row3item.text =@"去电影院看地心引力";
    _row3item.checked = NO;

    _row4item = [[ChecklistItem alloc]init];
    _row4item.text =@"看西甲巴萨新败的比赛回放";
    _row4item.checked = NO;


}
```

在之前的学习中我们做过类似的事情，但不同的是现在text和checked变量不再是当前视图控制器的实例变量，而是来自另一个世界的ChecklistItem对象的属性变量。

在设置具体的属性值之前，我们需要创建一个新的ChecklistItem 对象：

```
  _row0item = [[ChecklistItem alloc]init];
```

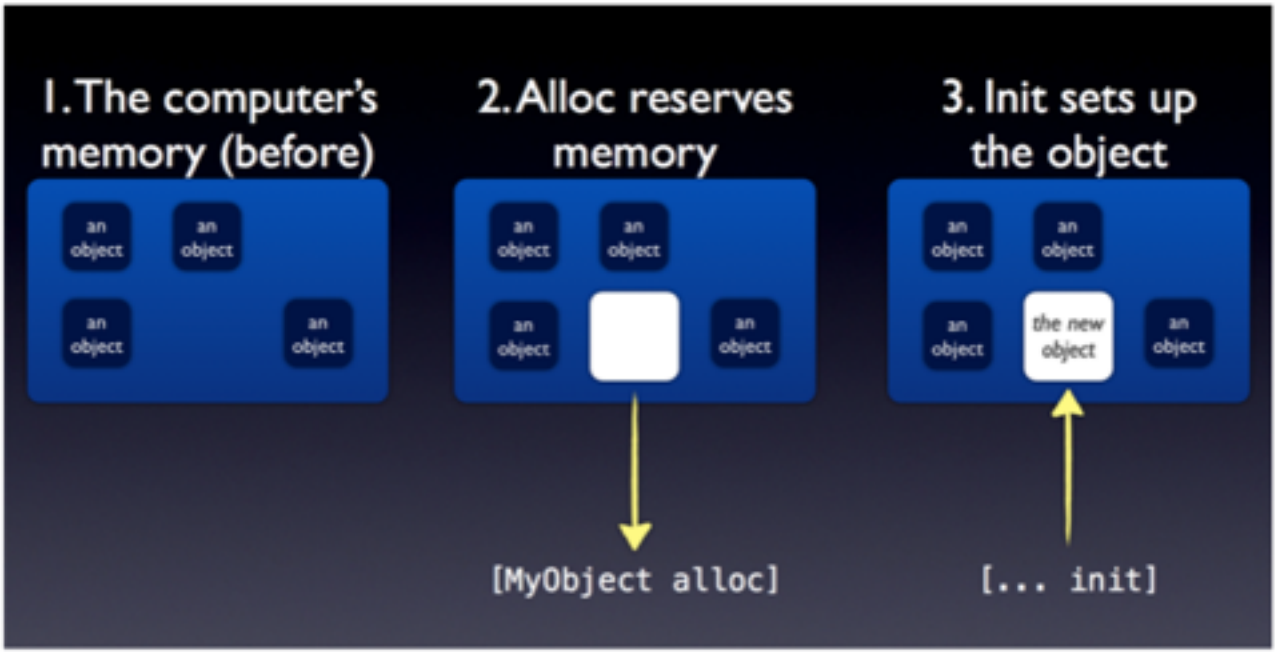
在头一篇教程中我们曾在创建UIAlertView提示框时用到过类似的方法：
alertView = [[UIAlertView alloc] initWithTitle:...];

在Objective-C中，创建新的对象就是这样完成的，首先我们调用alloc为新对象分配内存空间，然后用种种不同形式的init初始化方法来初始化对象。初始化就意味着这个对象现在可以被使用了，换句话说你可以给它们赋值了。

通常我们把这个过程称之为创建并初始化对象。

ChecklistItem对象只有一个默认的初始化方法是init，但并非所有的对象都这么简单化。比如UIAlertView的初始化方法就是
initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:
这么写是不是吓倒一大片？好吧，不要恐慌，don't panic，习惯了就好。
在iOS开发中，一个对象可能有多个init方法，但在实际使用的时候我们只能调用其中的一种。

好了，再次总结一下。每当我们需要创建一个新对象的时候都要走这个创建并初始化的流程。
先alloc，然后init。通过这个流程我们得到了对象的一个实例，也就是内存中对象的一份拷贝。术语党起了个足以吓尿一堆人的高大上名词-instantiation，确定你不会背错?



上图就是创建并初始化一个对象的过程。首先为该对象分配内存，然后用init方法来初始化一个对象（获得该对象的一个拷贝放到所分配的内存空间）。

在viewDidLoad方法中，创建完ChecklistItem对象后，我们就可以往里面填充text和checked属性了。注意我们为每一行都创建了一个单独的ChecklistItem对象。

接下来需要更改其它几个方法的代码：

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

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

        isChecked = _row0item.checked;

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

        isChecked = _row1item.checked;


    }if(indexPath.row ==2){
        isChecked = _row2item.checked;


    }if(indexPath.row ==3){

        isChecked = _row3item.checked;

    }if(indexPath.row ==4){
```

```
        isChecked = _row0item.checked;
    }
    if(isChecked){
        cell.accessoryType = UITableViewCellAccessoryCheckmark;
    }else{
        cell.accessoryType = UITableViewCellAccessoryNone;
    }

}

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

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

    if(indexPath.row  == 0){
        label.text = _row0item.text;
    }else if(indexPath.row  == 1){
        label.text = _row1item.text;
    }else if(indexPath.row  == 2){
        label.text = _row2item.text;
    }else if(indexPath.row  == 3){
        label.text = _row3item.text;
    }else if(indexPath.row  == 4){
        label.text = _row4item.text;
    }

    [self configureCheckmarkForCell:cell atIndexPath:indexPath];

    return cell;
}

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

    UITableViewCell *cell =[tableView cellForRowAtIndexPath:indexPath];


    if(indexPath.row ==0){

        _row0item.checked = !_row0item.checked;

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

         _row1item.checked = !_row1item.checked;

    }if(indexPath.row ==2){
         _row2item.checked = !_row2item.checked;
```

```
    }if(indexPath.row ==3){
        _row3item.checked = !_row3item.checked;

    }if(indexPath.row ==4){
        _row4item.checked = !_row4item.checked;
    }

    [self configureCheckmarkForCell:cell atIndexPath:indexPath];

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

以上代码看起来一大堆，其实改动很少，只不过用ChecklistItem对象的属性变量替代了之前所定义的实例变量。
就好比之前你直接在这里定义了几个实例变量：我的房产估值，我的黄金储量，我的现金，而现在用我的资产.房产估值，我的资产.黄金价值，我的资产.现金来替代了它们。道理是一样的，只是没那么直观。

如果你嫌麻烦，可以直接拷贝粘贴，但还是建议你一行行的敲出来比较好。
那啥卖油翁不是说过的吗，无它，唯手熟尔。日敲代码千百行，不会AB也会C。

为了确保你没犯什么低级错误，建议编译运行下，看看一切是否正常。

不过到目前为止，我们还是懒人的做法啊，传说中的数组也没有用到啊。
哦对，忘了顺便告诉你，数组也是数据结构的一种啊。

现在我们就来搞定这个东西！

继续之前，先学习一个小东西。

Mutable 和non-mutable

在iOS中有两种数组:mutable array（可变数组，也就是NSMutableArray)和non-mutable array（NSArray)。这里的可变指的是：可以改变。这不是废话吗？那么NSArray就是不可改变的。
什么叫不可改变呢？一旦你创建了一个NSArray对象，那么你不能往里面添加新的对象，也不能删除里面已有的对象，只能访问已经存在数组里面的对象。

类似的概念在iOS中还有很多。比如NSString是不可变的，还有一个NSMutableString就是可变的一旦我们创建了一个NSString对象，就不能直接更改其中的内容，而只能从它派生出一个新的具有不同内容的NSString对象。

例如，[string lowercase]会创建一个新的string对象，其中所有的字符都会被转换为小写。但原始的string对象仍然毫发无损。如果我们想修改一个string对象，最好的办法就是把它定义为NSMutableString类型。

不过需要注意的是，即便我们用的是non-mutalbe 数组，仍然可以修改其中的对象。不可变指的是数组本身不可变-我们不能从中删除已有的对象，或是加入新的对象，但原有对象的内容是可以改变的。只需要用类似array[objectAtIndex:]或array[index]的方式来获得某个对象的引用就可以放心大胆的修改对象内容了。

或许你对non-mutable 数组还是有点难以理解。
那还是打个比方吧，不列颠帝国的卡梅隆战士昨日拜访尘都，并勇敢尝试了这边的一处知名火锅。
据说老板要推出相同菜式的首相套餐，也就是7素2荤共9个菜式，连菜名都不变的，火锅锅底显然也必须是相同的。
但同是首相套餐，同名菜式，每个菜分量的多少却是可以酌情考虑的。至于蘸碟用的油吗，当然首相不可能体验到我天朝特产的中地沟出品的油，至于屌丝屁民们就说不准了。
你懂了吗？你懂的。

当然，这里不是要和店家过不去，香天下我还是去过的，就在会展中心靠近天鹅湖旁边，环境相当不错，价格略贵，味道尚可，值得一去。

不过这里我们需要用到一个可变数组，因为在当前这款应用中需要让用户添加和删除新的项目，就好比有人凑热闹但又要吃定制版可增减菜品的首相套餐，而不是原版的。

回到Xcode,切换到ChecklistViewController.m，删除之前的实例变量声明，使用一个单一的NSMutableArray变量来替代：

```
@implementation ChecklistsViewController{

    NSMutableArray *_items;

}
```

更改viewDidLoad方法的代码如下：

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


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

    ChecklistItem *item;

    item = [[ChecklistItem alloc]init];
    item.text =@"观看嫦娥飞天和玉兔升空的视频";
    item.checked = NO;
    [_items addObject:item];

    item = [[ChecklistItem alloc]init];
    item.text =@"了解Sony a7和MBP的最新价格";
    item.checked = NO;
    [_items addObject:item];

    item = [[ChecklistItem alloc]init];
    item.text =@"复习苍老师的经典视频教程";
    item.checked = NO;
    [_items addObject:item];
```

```
    item = [[ChecklistItem alloc]init];
    item.text =@"去电影院看地心引力";
    item.checked = NO;
    [_items addObject:item];

    item = [[ChecklistItem alloc]init];
    item.text =@"看西甲巴萨新败的比赛回放";
    item.checked = NO;
    [_items addObject:item];


}
```

和之前的代码区别不大，不同的是我们用到了数组对象：

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

好吧，又见[[alloc]init...]模式用于创建并初始化一个对象。
记住，声明一个变量并不意味着你就自动创建了一个该类型的对象。变量仅仅是对象的容器，我们仍然需要用alloc和init来创建对象并将其保存在变量中。

又比如下面的代码：

```
@implementation ChecklistsViewController{

    NSMutableArray *_items;

}
```

这里只是说：我有一个名为_items的变量，它是NSMutableArray类型的。但直到你创建并初始化一个真正的NSMutableArray对象，并将其保存到_items前，变量是空的。用程序猿的话叫nil，或者某些猿类喜欢叫null。在iOS中我们可以给nil变量发送消息，但最终这些消息不知道会跑哪儿去了，这样做毫无意义。

因此在viewDidLoad方法中，我们首先创建并初始化了一个NSMutableArray对象，并将其保存在_items变量中。
NSMutableArray对象有一个init方法名为initWithCapacity:，其作用是初始化一个对象，并预留一个特定数字的项目空间（比如这里是20）。不过请注意，这里并不是说这个数组只能保存20个项目！！！这不过是我们给数组的一个提示，如果需要超过20个项目，只需要往里面添加就是了，数组会自动扩展空间以满足需求。

然后我们会看到，每次我们创建了一个ChecklistItem对象时，都会将其添加到数组中：

```
    item = [[ChecklistItem alloc]init];
    item.text =@"观看嫦娥飞天和玉兔升空的视频";
    item.checked = NO;
    [_items addObject:item];
```

在viewDidLoad方法的最后，_items数组已经包含了5个ChecklistItem对象，这就是我们的新数据模型。

现在我们的数组里面已经包含了所有的行，接下来可以简化便试图的数据源和代理方法了。

更改以下方法的代码：

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


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


    ChecklistItem *item = _items[indexPath.row];

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

}


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

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


    ChecklistItem *item = _items[indexPath.row];

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

    label.text = item.text;
    [self configureCheckmarkForCell:cell atIndexPath:indexPath];

    return cell;
}

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

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

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

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

在以上的每个方法中，我们都做了相同的事情：
```
  ChecklistItem *item = _items[indexPath.row];
```

使用这行代码，我们从数组中请求具备和行编号相同的index的ChecklistItem对象。
一旦获取了该对象，我们就可以查看其text和checked属性，并做自己想做的事情了。即便用户要添加100个以上的代办事务清单，上述方法的代码也不会发生改变。

注意：
还有一种从数组中获取对象的方式：
```
ChecklistItem *item = [_items objectAtIndex:indexPath.row];
```

在N年的传统中，以上方法是Objective-C中获取数组中对象的唯一方法，直到最近才添加了我们现在用的这种新方式。如果你之前用过其它语言，对此会感激流涕的。

还有一处小地方需要修改，我们最好用数组中对象的数目替代硬编码的一个数值。

更改tableView:numberOfRowsInSection:方法的内容：

```
-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{
    return [_items count];
}
```

这样一来，不但代码显得清晰易读，而且可以处理任意数量的行。这就是array数组的威力。

编译运行项目，看看效果吧。当然实际上没有任何变化，只是整个代码结构更加清晰易读

小练习：
试着添加几行新数据，现在你只需要修改viewDidLoad方法就行了。

好了，今天的学习到此结束，又到了派发福利的时候了。

首先是卡梅伦昨天涮火锅的抵价券，当然不可能白送了。

http://t.dianping.com/deal/5148358

然后是杜甫草堂的茶和妹子压阵