从零开始学iOS7开发系列教程-事务管理软件开发实战-Chapter23

开发环境：
Xcode 5 +iOS 7

欢迎继续我们的学习。

现在可以继续来完善这个应用。
在使用列表的时候，经常要做的一件事情就是根据某个特定的顺序对其中的项目进行排序。这里我们将根据事项名称对列表进行排序。

现在每当我们添加一个新的checklist，都会自动添加到列表的最后。

在我们学习如何来对一个数组进行排列前，先想想看何时需要对数组进行排序：

1.当用户添加一个新的checklist时
2.当某个checklist被重命名的时候

此外，当某个checklist被删除的时候就没有必要重新排序了，因为它不会对现有对象的顺序有任何影响。

对于这两种情况的处理目前是在AllListsViewController的didFinishAddingChecklist和didFinishEditingChecklist方法中实现的。

打开Xcode,切换到AllListsViewController.m，对以上两个方法的代码更改如下：

```
- (void)listDetailViewController:(ListDetailViewController *)controller didFinishAddingChecklist:(Checklist *)checklist
{
    [self.dataModel.lists addObject:checklist];
    [self.dataModel sortChecklists];
    [self.tableView reloadData];
    [self dismissViewControllerAnimated:YES completion:nil];
}

- (void)listDetailViewController:(ListDetailViewController *)controller didFinishEditingChecklist:(Checklist *)checklist
{
    [self.dataModel sortChecklists];
    [self.tableView reloadData];

  [self dismissViewControllerAnimated:YES completion:nil];
}
```

之所以这里可以删掉一大段的代码，是因为我们可以用reloadData方法让表视图重新加载数据。我们不再需要手动插入新的行，也不需要手动更新cell的textLabel信息。我们只需要简单调用reloadData方法就可以更新整个表的内容。

别害怕效率问题，因为在AllListsViewController的表视图中不太可能有过多的行。如果表视图中需要显示成百上千行，或许我们需要考虑换一种方式来实现。

不过这里出现了一个新的sortChecklists方法，让我们来添加下它的声明和实现代码：

切换到DataModel.h，添加一个方法声明：
```
-(void)sortChecklists;
```

然后切换到DataModel.m，添加该方法的实现代码：

```
-(void)sortChecklists{

    [self.lists sortUsingSelector:@selector(compare:)];

}
```

self.lists属性的数据类型是NSMutableArray。该对象有一个非常方便易用的sortUsingSelector方法。selector其实是方法的名称。这里我们告诉lists数组，它需要使用compare:方法来对内容进行排序。该方法并非是数组本身所拥有的，而是它所包含的对象-Checklists所拥有的。

接下来排序算法会调用[Checklist compare]来检查Checklist对象间的关系。遗憾的是该排序算法实际上对Checklist对象还一无所知，因此我们需要帮它一把。

切换到Checklist.m，然后添加一个compare方法：
```
-(NSComparisonResult)compare:(Checklist*)otherChecklist{

    return [self.name localizedStandardCompare:otherChecklist.name];
}
```

或许你以为这个比较会很复杂，其实只需要上面方法中的一行代码就行了。为了比较两个Checklist对象，我们只需要比较当前Checklist对象和otherChecklist对象的名称差异。因为name属性属于NSString，所以它有一个很方便的方法可用，那就是localizedStandardCompare。

该方法会比较两个name对象，并忽略大写和小写的区别（a和A会被认为是相同的），同时还会考虑所在地区的规则。locale是一个对象，它会了解用户所在的国家和语法的特殊规则。比如在德语中的排序和在英文中会有所区别。

有了这个compare:方法，NSMutableArray的sortWithSelector方法就会重复询问某个Checklist对象，它和另一个Checklist对象的比较结果是怎样的，然后根据比较结果来调整它们的顺序，直到整个数组排序完成。在Checklist对象的compare方法中，我们简单比较 了两个对象的name属性。如果我们想基于其它要素进行排序，只需要更改这个compare方法就好了。

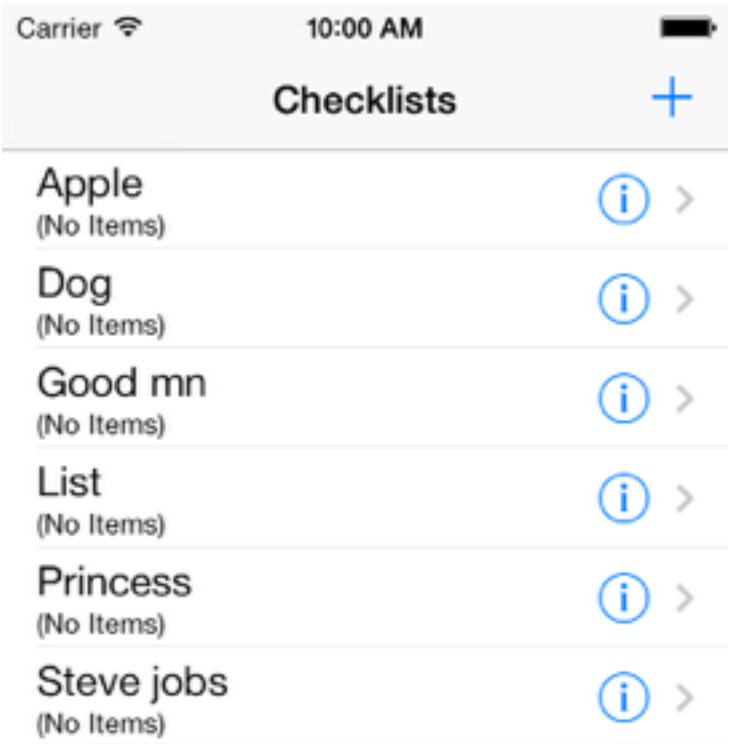
理论充电- 关于如何使用selector（动态方法名称解决方案）

或许你还在想，为什么这里我们没有在Checklist.h中添加compare:方法的声明？因为我们用到了一个小奥义，可以节省掉这个工作。

如果我们在代码中直接调用[checklist compare]方法，那么就肯定需要在Checklist.h文件中声明这个方法。不过这里我们用到了selector，它可以在runtime（运行时，也即当应用在Simulator火设备中运行的时候）解决方法的名称，而不是在编译时就提供。

对于这种动态方法名称解决方案，我们无需在.h中添加方法声明。不过这也导致了可能的危险：你很有可能在某个对象上调用一个根本不存在的selector。
这就是为什么我们经常在应用崩溃时看到”unrecognized selector sent to instance xxx”错误消息的原因。

好了，现在可以编译运行应用，然后添加一些checklists。更改它们的名称，然后可以看到checklist的列表会根据字母顺序自动排序。

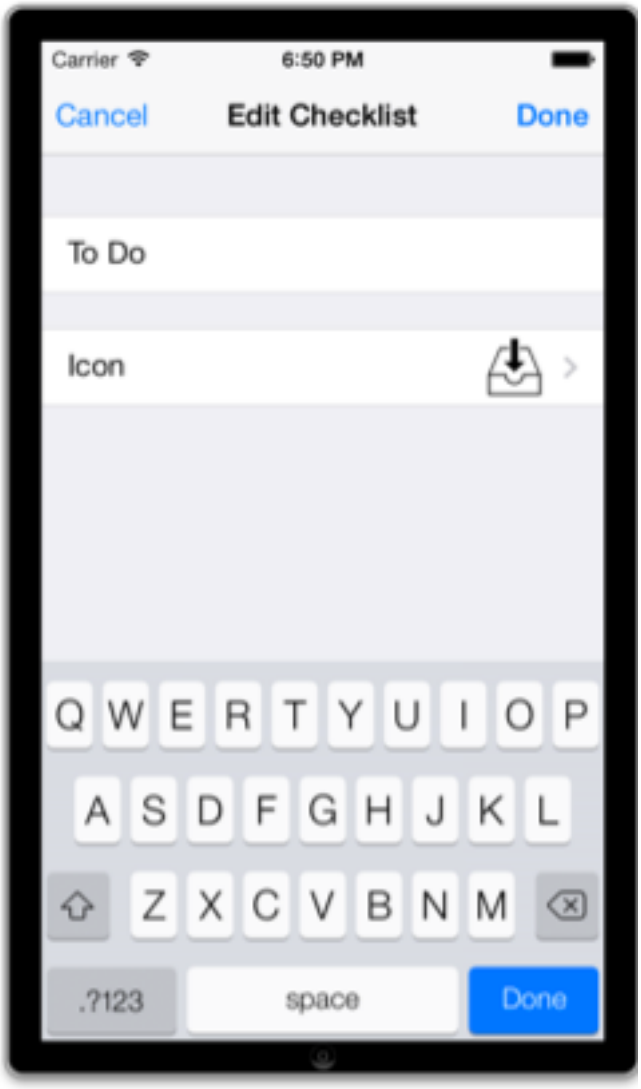


看，这样我们将用户体验又提升了一个小小的级别。

接下来还是细节调整。

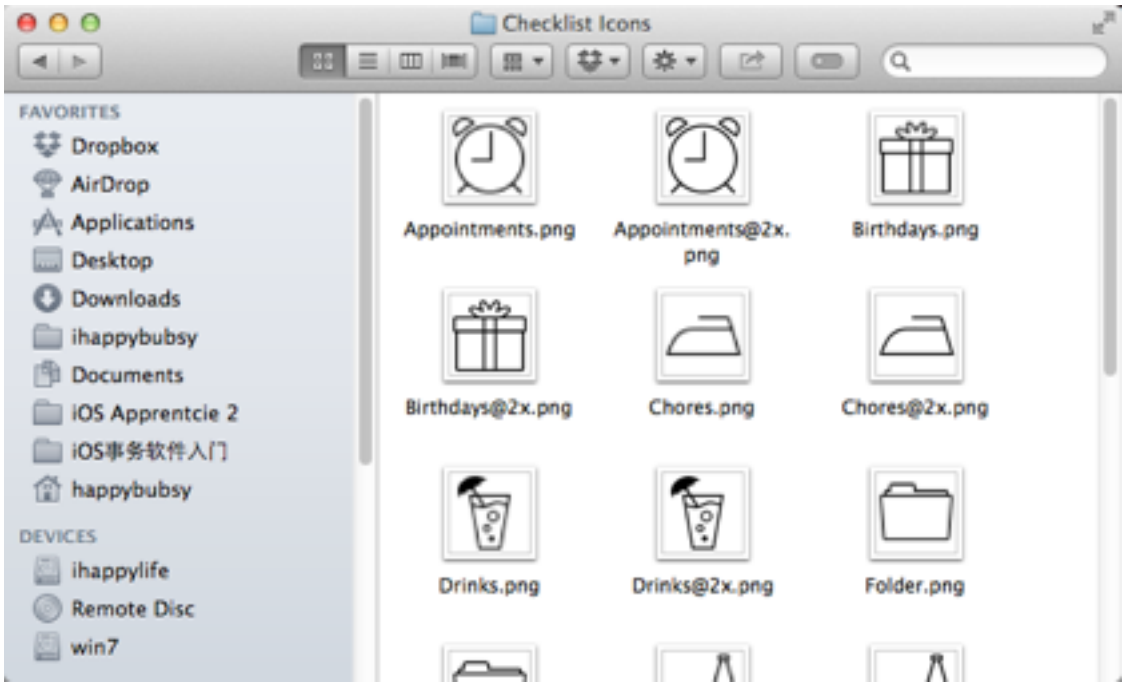
让我们给checklist添加图标，从而可以给用户更图形化和直观的感受。当然，除了可以改善用户体验之外，通过学习如何给checklist添加图标，还可以再重温下视图控制器和代理协议的知识。这些知识实在是太重要了，我真心希望你即便在梦里面也知道它们是肿么回事。
让我们对Checklist对象添加新的设置，从而可以让用户选择checklist的图标。

当完成这个任务后，Add/Edit Checklist界面将会是下面的样子：

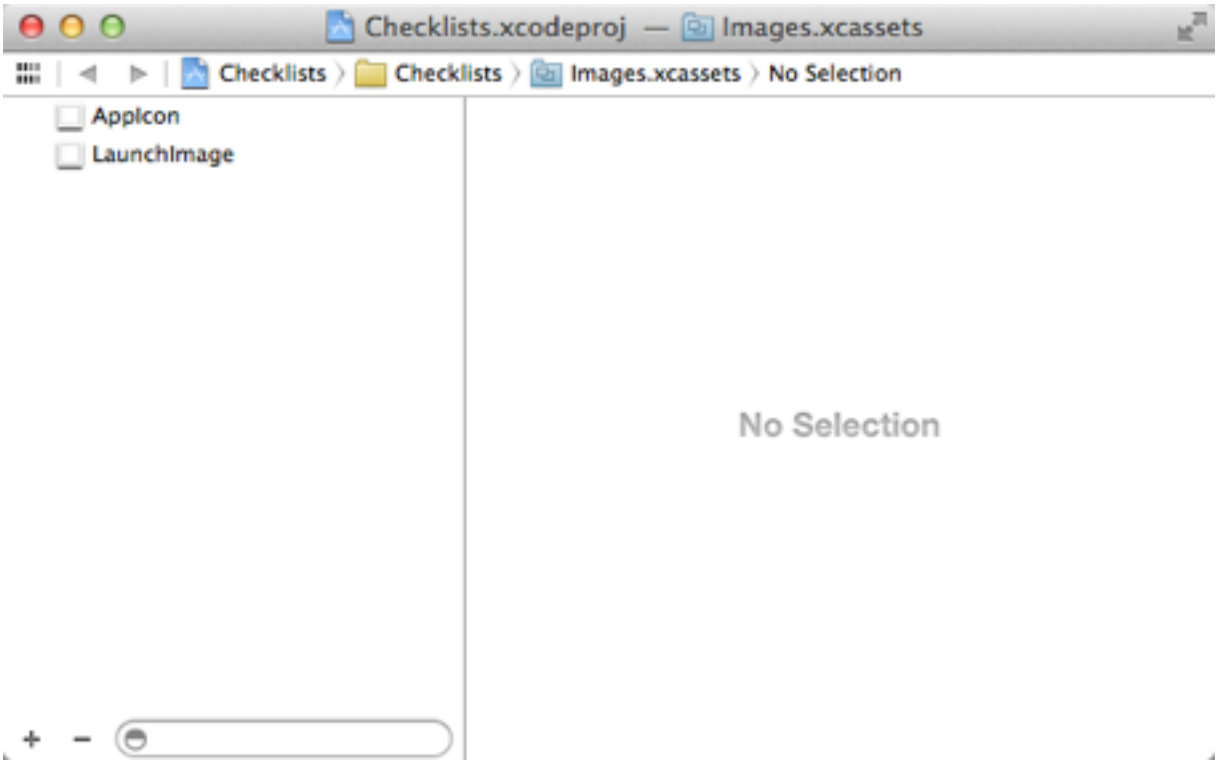


我们将在Add/Edit Checklist界面上添加一行，可以让用户打开一个新的界面来选择图标。这里用来选择图标的视图控制器将是一个新的视图控制器类型。我们不会用modal的方式来切换界面，而是把它push到导航视图层级堆栈上。

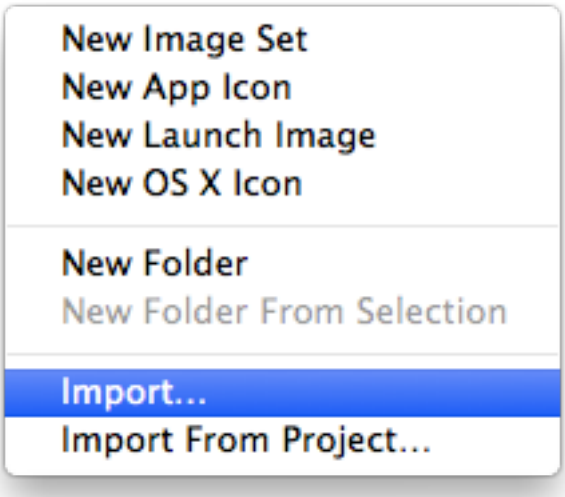
在本章的示例代码中，你回看到在Resources文件夹下面有一个名为Checklist Icons的文件夹，其中有一些PNG图片。当然，这些图片看起来平淡无奇，实际开发应用的时候你可能需要换成自己亲手设计的精美图标。



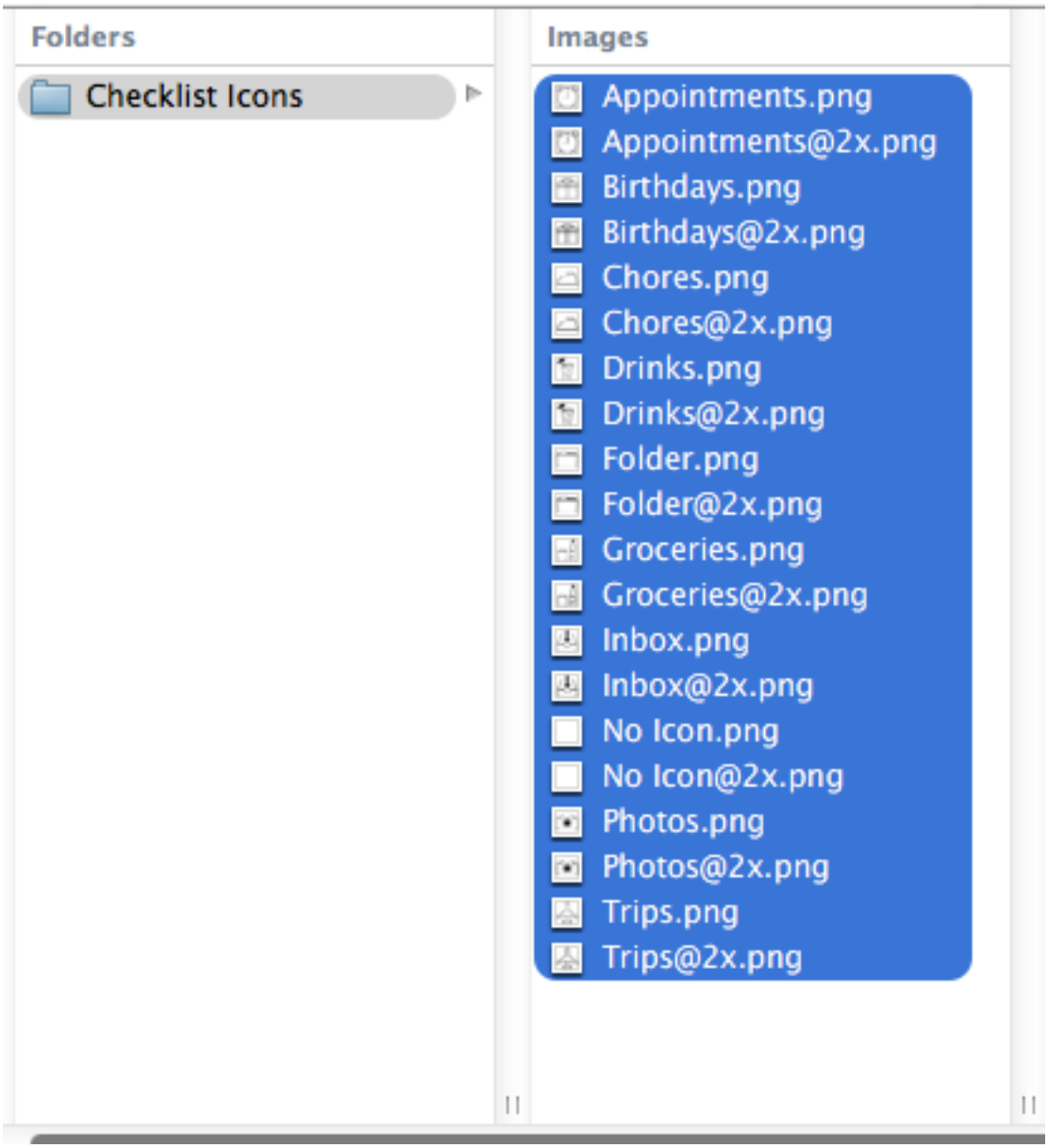
将这些图片添加了项目的asset中。
从Xcode左侧的项目导航面板中双击Image.xcassets，可以看到下面的画面：



然后点+按钮，选择Import...

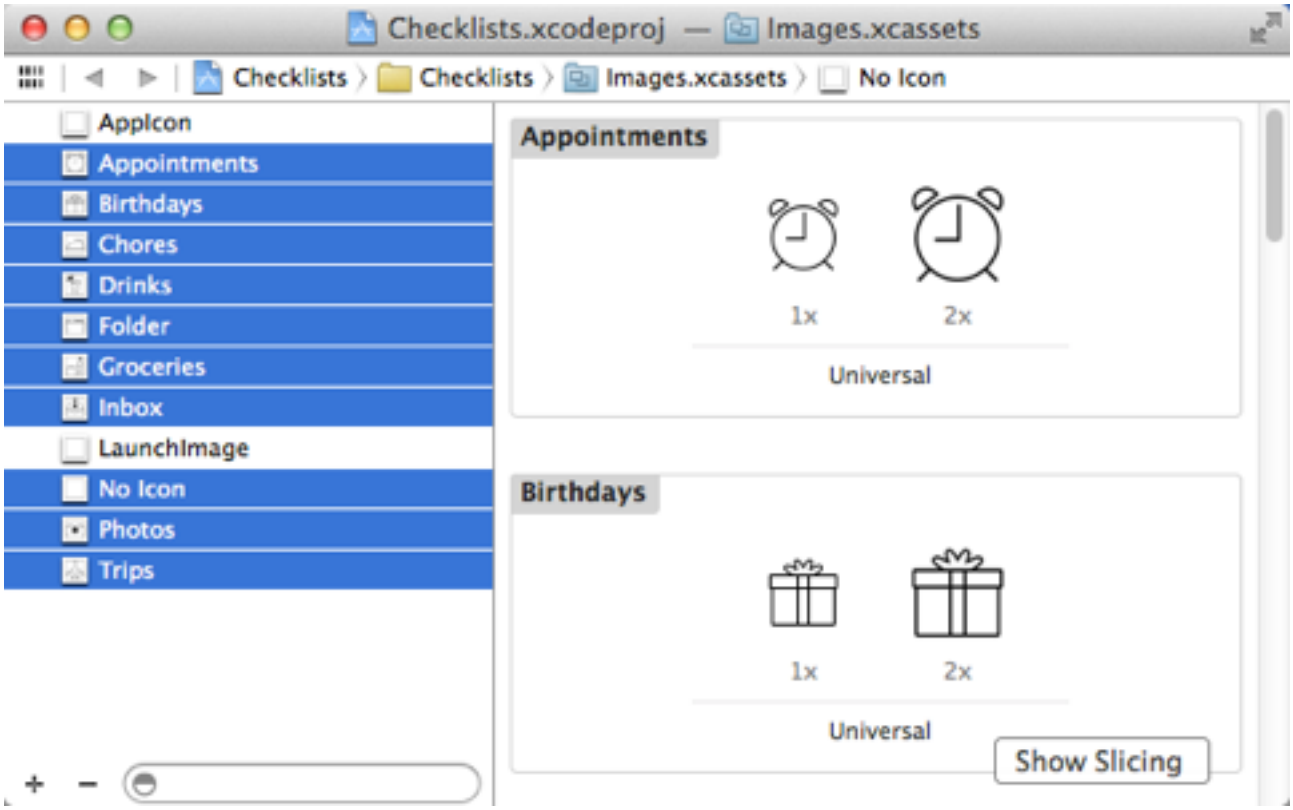


切换到刚才的Checklist Icons文件夹，并选中其中的所有文件：



注意：选中所有的文件，而非文件夹。

然后点击Open，此时可以看到这些资源都已经添加到项目的asset里面了。



注意里面的所有资源都有1x版本和2x版本，其中1x版本是针对低分辨率设备的，而2x版本是针对retina分辨率设备的。

不过根据苹果的最新规定，所有针对iPhone的iOS应用都必须提供高分辨率的retina图片。因此1x版本图片的唯一用途就是，假如你的应用要支持iPad（比如目前仍然很火热的iPad 2），那么可以考虑提供。否则的话，还是替美术和设计人员减轻点负担吧。

在Xcode中切换到Checklist.h，然后添加属性声明如下：

```
@property(nonatomic,copy)NSString *iconName;
```

然后切换到Checklist.m，更改其中的initWithCoder和encodeWithCoder方法如下：

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

    if((self =[super init])){
        self.name = [aDecoder decodeObjectForKey:@"Name"];
        self.items = [aDecoder decodeObjectForKey:@"Items"];
```

```
    self.iconName = [aDecoder decodeObjectForKey:@"IconName"];
  }
  return self;

}

-(void)encodeWithCoder:(NSCoder *)aCoder{


  [aCoder encodeObject:self.name forKey:@"Name"];

  [aCoder encodeObject:self.items forKey:@"Items"];

  [aCoder encodeObject:self.iconName forKey:@"IconName"];
}
```

通过黄色高亮部分代码，我们就可以在Checklists.plist文件中保存icon名称了。

需要注意的是，当你开发自己的应用时，每当增加一个新的属性，都需要做类似上面的工作，否则是不会被保存到plist文件中的。

为了测试下代码，让我们顺便也更改下init方法：

```
-(id)init{

  if((self = [super init])){
    self.items = [[NSMutableArray alloc]initWithCapacity:20];
    self.iconName = @"Appointments";
  }
  return self;
}
```

这样所有的checklist都有了Appointments这个图标。此时我们只想确认可以为checklist添加一个图标（任何图标），并且在表视图中显示。当实现这个小小的进步后，再来考虑如何让用户可以选择自己想要的图标。

在Xcode中切换到AllListsViewController.m，更改cellForRowAtIndex方法的代码如下：

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
 static NSString *CellIdentifier = @"Cell";

 UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
 if (cell == nil) {
  cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
reuseIdentifier:CellIdentifier];
 }
```

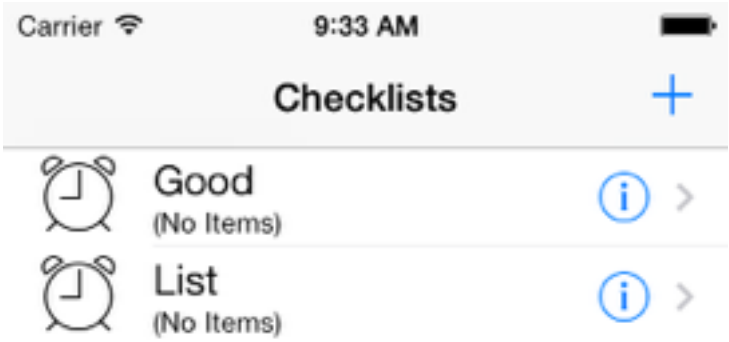
```
  Checklist *checklist = self.dataModel.lists[indexPath.row];

  cell.textLabel.text = checklist.name;
  cell.accessoryType = UITableViewCellAccessoryDetailDisclosureButton;

    int count = [checklist countUncheckedItems];
    if([checklist.items count]==0){

        cell.detailTextLabel.text = @"(No Items)";
    }else if(count ==0){
        cell.detailTextLabel.text =@"全部搞定收工！";
    }else{
  cell.detailTextLabel.text = [NSString stringWithFormat:@"%d Remaining",[checklist
countUncheckedItems]];
    }

    cell.imageView.image = [UIImage imageNamed:checklist.iconName];


  return cell;
}
```

使用UITableViewCellStyleSubtitle样式的cell在左侧有一个内置的UIImageView。我们只需要指定一个图片给它，就会自动显示出来。

好了，现在可以来看卡我们的成果了。

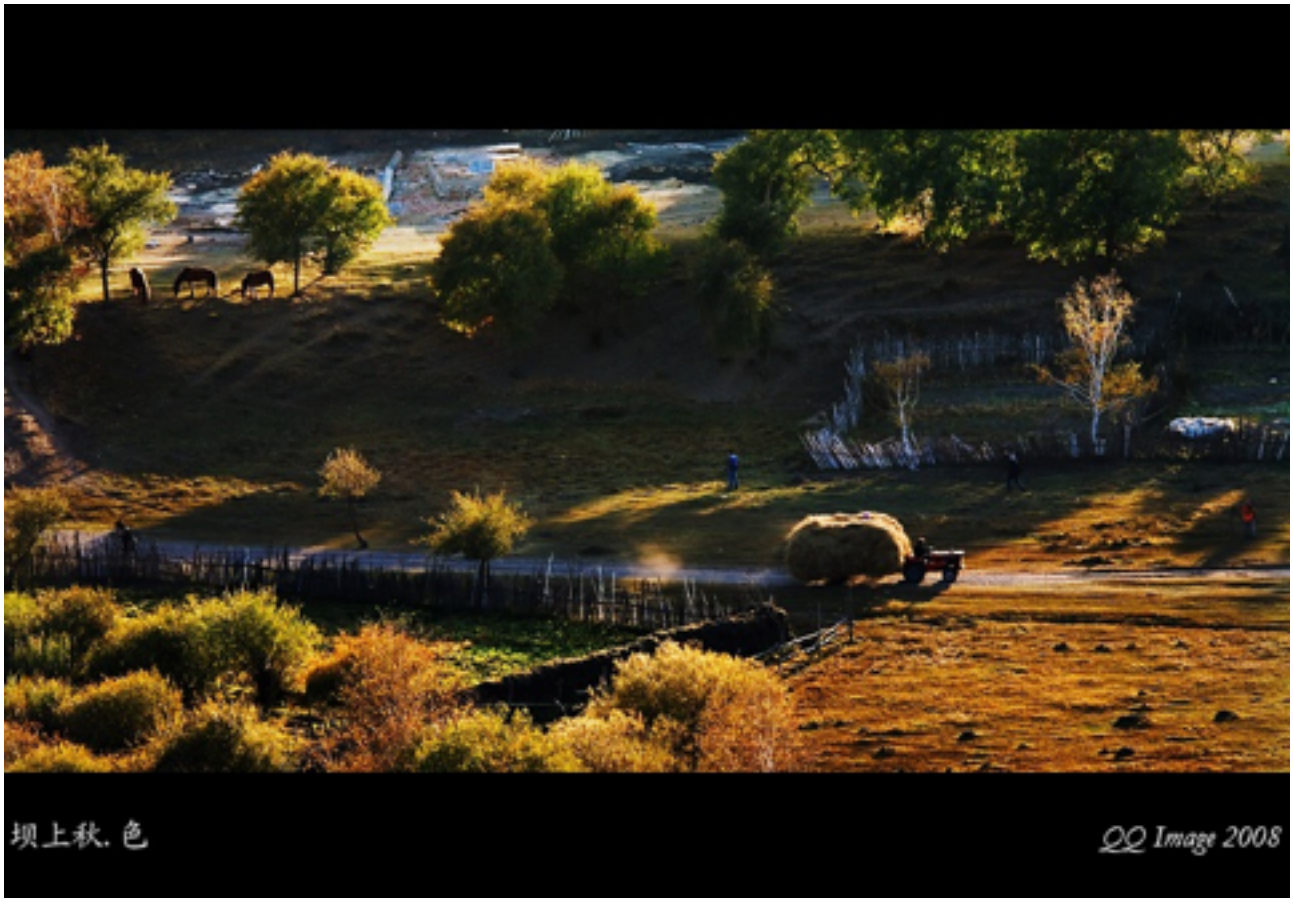
不过在编译运行应用之前，记得要删除沙盒中的Checklists.plist文件，或者reset下Simulator，因为我们再次修改了数据模型（在init/encodeWithCoder中添加了一个IconName)。否则的话你回被莫名其妙的应用崩溃搞得不知所措的~

编译运行应用，可以看到每个checklist现在都将拥有自己的图标了。



好了，该休息一下，先欣赏下今天的福利吧。

在雾霾漫天的世界里，是不是在梦中想过这样的踏春场景呢?



美景尚需美女陪衬