iOS 5 Programming Cookbook

第三章

构造和使用 TableView

版本 1.0

翻译时间：2012-05-28

DevDiv 翻译： kyelup cloudhsu 耐心摩卡
wangli2003j3 xiebaochun
DevDiv 校对： laigb kyelup
DevDiv 编辑： BeyondVincent

3244424530434

写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问 DevDiv 移动开发论坛 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

关于本文的翻译

感谢 kyelup、cloudhsu、耐心摩卡、wangli2003j3 和 xiebaochun 对本文的翻译，同时非常感谢 laigb 和 kyelup 在百忙中抽出时间对翻译初稿的认真校验，指出了文章中的错误。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

读者查看下面的帖子可以持续关注章节翻译更新情况

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译各章节汇总

目录

写在前面 2
关于 DevDiv 2
技术支持 2
关于本文的翻译 2
目录 4
前言 6
第 1 章 基础入门 7
第 2 章 使用控制器和视图 8
第 3 章 构造和使用 Table View 9
3.0. 介绍 9
3.1. 实例化 Table View 9
3.1.1. 问题 9
3.1.2. 方案 9
3.1.3. 讨论 9
3.1.4. 参考 10
3.2. 将委托分配到 Table View 11
3.2.1. 问题 11
3.2.2. 方案 11
3.2.3. 讨论 11
3.2.4. 参考 12
3.3. 向 Table View 填充数据 12
3.3.1. 问题 12
3.3.2. 方案 12
3.3.3. 讨论 12
3.3.4. 参考 15
3.4. 接收和处理 Table View 事件 16
3.4.1. 问题 16
3.4.2. 方案 16
3.4.3. 讨论 16
3.4.4. 参考 17
3.5. 在 Table View 中使用不同种类的附件 17
3.5.1. 问题 17
3.5.2. 方案 17
3.5.3. 讨论 18
3.5.4. 参考 19
3.6. 创建自定义 Table View 单元格附件 19
3.6.1. 问题 19
3.6.2. 方案 20
3.6.3. 讨论 21
3.6.4. 参考 22

3.7. 在 Table View 中展示分层数据 22
3.7.1. 问题 22
3.7.2. 方案 22
3.7.3. 讨论 23
3.7.4. 参考 23
3.8. 启用 Table Viewcell 的滑动删除 24
3.8.1. 问题 24
3.8.2. 方案 24
3.8.3. 讨论 25
3.8.4. 参考 25
3.9. 在 Table View 中构建页眉和页脚 26
3.9.1. 问题 26
3.9.2. 方案 26
3.9.3. 讨论 26
3.9.4. 参考 34
3.10. 在 Table Viewscell 中展示快捷菜单 34
3.10.1. 问题 34
3.10.2. 方案 34
3.10.3. 讨论 34
3.10.4. 参考 37
3.11. 在 Table View 中移动 cell 和 Section 38
3.11.1. 问题 38
3.11.2. 方案 38
3.11.3. 讨论 38
3.11.4. 参考 43
3.12. 从 Table View 删除单元格和 Section 44
3.12.1. 问题 44
3.12.2. 方案 44
3.12.3. 讨论 44
3.12.4. 参考 44

前言

参考帖子
[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_前言

第 1 章 基础入门

参考帖子

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第一章_基础入门

第 2 章 使用控制器和视图

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译 第二章 使用控制器和视图(上)

[DevDiv 翻译]iOS 5 Programming Cookbook 翻译_第二章_使用控制器和视图(下)

第 3 章　构造和使用 Table View

3.0. 介绍

简而言之，Table View 是一个被分成不同部分的滚动视图，每部分又进一步被分成行。每行是一个 UITableViewCell 类的实例。通过此类的子类，可以创建自定义 Table View 的行。使用 Table View 是一个向用户展示项目列表的理想方法。你可以把图片、文本和其他任何东西嵌入 Table View 单元格，可以自定义它们的高度、形状、分组或更多。Table View 结构的简易性使其高度可自定义。一个 Table View 可以输入 Table View 数据源的数据，你会接收到各种事件并用 Table View 委托对象来控制其物理外观。这些分别在 UITableViewDataSource 和 UITableViewDelegate 的协议来定义。尽管 UITableView 实例将 UIScrollView 子类化，Table View 只能垂直滚动；这更大程度上是一个特点而非局限。本章中，我们将讨论创建、管理和自定义 Table View 的不同方法。

3.1. 实例化 Table View

搭建开发环境是进行软件开发的前提，一般手机操作系统软件供应商都会为我们提供比较完备的开发环境和软件开发包。

3.1.1. 问题

在你的 UI 中放置一个 Table View。

3.1.2. 方案

实例化一个 UITableView 类的对象并把它做为一个子视图添加到你的视图中。

3.1.3. 讨论

有 2 个办法可以实例化 Table View：

1. 通过代码
2. 使用界面生成器

如果你使用界面生成器,创建一个 Table View 就是简单的把一个 Table View 从对象库拖拽到你的.xib 文件中。如果你使用代码创建组建更顺手的话，也没有问题。你必须做的事情就是把 UITableView 类的一个对象实例化，让我们通过定义在视图控制器的头文件中定义我们的 Table View 开始：

```
#import <UIKit/UIKit.h>
@interface Instantiating_a_Table_ViewViewController : UIViewController
@property (nonatomic, strong) UITableView *myTableView;
@end
```

然后我们会继续并在视图控制器的实现文件中合成我们的 Table View：

```
#import "Instantiating_a_Table_ViewViewController.h"
@implementation Instantiating_a_Table_ViewViewController
@synthesize myTableView;
```

创建视图控制器就像分配和初始化 UITableView 的实例一样简单：

```
- (void)viewDidLoad{
    [super viewDidLoad];

    self.view.backgroundColor = [UIColor whiteColor];

    self.myTableView =
        [[UITableView alloc] initWithFrame:self.view.bounds
style:UITableViewStylePlain];

    [self.view addSubview:self.myTableView];

}
```

视图控制器中初始值设定项 initWithFrame:style:的样式参数允许我们指定我们需要哪种类型的 Table View。有 2 种风格我们可以选择：

UITableViewStylePlain

创建一个没有背景图片的空白 Table View

UITableViewStyleGrouped

创建一个有背景图片和圆角组边框的 Table View，类似于 Settings app。

如果你现在在 iPhone 模拟器上运行 APP，将看到没有填充 Table View 单元格的 Table View 放在那里：

图 3-1. 一张没有内容的空白 Table View

3.1.4. 参考

XXX

3.2. 将委托分配到 Table View

3.2.1. 问题

决定要把一个委托分配到 Table View。

3.2.2. 方案

将遵循 UITableViewDelegate 协议的对象指定为 Table View 的委托属性：

```
- (void)viewDidLoad{
   [super viewDidLoad];

   /* We want a full-screen Table View which is as
big as the View which is attached to the current
    View Controller */
   CGRect tableViewFrame = self.view.bounds;

   self.myTableView = [[UITableView alloc]
initWithFrame:tableViewFrame
style:UITableViewStylePlain];

   self.myTableView.delegate = self;

   /* Add this Table View to our View */
   [self.view addSubview:self.myTableView];

}
```

突出显示的语句将当前对象指定为 Table View 的委托。myTableView 是属于调用视图控制器的 UITableView 中的一个属性。该语句嵌入在 viewDidLoad 方法中，因为调用对象继承了 UIViewController 的实例；由于这个方法是放置语句的最好位置，所以关联可以一次完成。

3.2.3. 讨论

UITableView 类定义调用委托的属性，Table View 可以把这个属性分配给一个遵循 UITableViewDelegate 协议的对象；换句话说，这个委托必须保证回应此协议中定义的消息，这些消息通过 Table View 本身发送到委托对象。把 Table View 的委托看作是收听由 Table View 发送各种事件的对象，例如当一个单元格被选中，或者当一个 Table View 要指出它的单元格的高度。某种程度上，我们也可以使用 Interface Builder 来修正 Table View 及其单元格的视图外观。只要打开 Interface Builder，选择一个你之前创建的 Table View，然后选择 Tools→Size Inspector；在 Size Inspector 板，可以通过改变 Table View 单元格的的高度值就可以修改 Table View 的视觉外观。为了使你选择的 Table View 的委托对象符合 UITableViewDelegate 协议，你需要通过下面的办法把协议增加到对象的接口声明中：

```
#import <UIKit/UIKit.h>
@interface Assigning_a_Delegate_to_a_Table_ViewViewController
                : UIViewController <UITableViewDelegate>
@property (nonatomic, strong) UITableView *myTableView;
@end
```

对于那些 UITableViewDelegate 协议带有@required 标记的消息要强制委托对象回应。可以选择性的回应其他消息，但是委托必须回应任何一条你想影响 Table View 的消息。

发送到一个 Table View 的委托对象的消息将携带一个参数，这个参数会告诉委托对象哪个 Table View 发出的消息；注意到这一点非常重要，因为在特定情况下你需要在一个对象（通常是视图）中放置超过一张 Table View。因为这一点，极力推荐你做出的决定是基于事实上向你的委托对象发送确切消息的 Table View，比如：

```
- (CGFloat)        tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath{

   CGFloat result = 20.0f;

if ([tableView isEqual:self.myTableView]){
result = 40.0f;
   }

return result;

}
```

值得注意的是 Table View 中单元格的位置由其索引路径展示出来。一个索引路径是 section 和 row 索引的组合；section 索引是从零开始的并指定每个单元格属于哪个分组或 Section，而在 section 中的单元格索引又是从零开始的。

3.2.4. 参考

方法 3.4

3.3. 向 Table View 填充数据

3.3.1. 问题

你想要在 Table View 中填充数据。

3.3.2. 方案

在一个对象中遵循 UITableViewDataSource 协议，并将该对象指定为表的视图的数据源属性。

3.3.3. 讨论

创建一个遵循 UITableViewDataSource 协议的对象并把它分配到一个 Table View 实例中；然后通过响应数据源消息给你的 Table View 提供信息。举个例子，我们来看下，在我们的视图控制器中声明.h 文件，这个文件随后会在其视图中创建一个 Table View，看下代码：

```
#import <UIKit/UIKit.h>
@interface Populating_a_Table_View_with_DataViewController
                : UIViewController <UITableViewDataSource>
```

```
@property (nonatomic, strong) UITableView *myTableView;
@end
```

现在让我们来合成 Table View:

```
#import "Populating_a_Table_View_with_DataViewController.h"
@implementation Populating_a_Table_View_with_DataViewController
@synthesize myTableView;
```

在我们的视图控制器中用 viewDidLoad 方法可以创建 Table View，同时把我们的视图控制器按照其数据源分配出去：

```
- (void)viewDidLoad{
   [super viewDidLoad];

   self.view.backgroundColor = [UIColor whiteColor];

   self.myTableView =
      [[UITableView alloc] initWithFrame:self.view.bounds
style:UITableViewStylePlain];

   self.myTableView.dataSource = self;

   /* Make sure our table view resizes correctly */
   self.myTableView.autoresizingMask =
      UIViewAutoresizingFlexibleWidth |
      UIViewAutoresizingFlexibleHeight;

   [self.view addSubview:self.myTableView];
}
```

现在我们需要确定我们的表视图响应了 UITableViewDataSource 协议的必要方法，按住键盘上的 command 键，然后连接到有声明 UITableViewDataSource 协议的视图控制器头文件中，这将会展示给你这个协议必要的方法。

UITableView 类定义了一个调用 dataSource 的属性，这个非类型化的对象必须遵循 UITableViewDataSource 协议，每次刷新表格视图，并重新加载使用 reloadData 方法时这个 Table View 会从其数据源调用各种方法以找出你打算对其填充的数据，一个 Table View 的数据源能够执行三个重要方法，其中 2 个对每个数据源都要强制执行：

numberOfSectionsInTableView：

此方法允许数据源告知必须加载到 Table View 中的表的 Section 数。

tableView:numberOfRowsInSection:

此方法告诉视图控制器有多少单元格或者行要下载到每个 Section，Section 个数传递给数据源中的 numberOfRowsInSection 作参数，这个方法在数据源对象中要强制执行。

tableView:cellForRowAtIndexPath:

此方法负责返回作为 Table View 行的 UITableViewCell 类静态实例。这个方法在数据源对象的执行中也是强制性的。所以我们来在视图控制器中执行逐个执行这些方法。首先，我们告诉 Table View 我们要它呈现 3Section：

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{

   NSInteger result = 0;
if ([tableView isEqual:self.myTableView]){
result = 3;
```

```
    }
return result;

}
```

然后我们告诉表视图需要它在每个 Section 呈现多少行：

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{

    NSInteger result = 0;
if ([tableView isEqual:self.myTableView]){
switch (section){
case 0:{
result = 3;
break;
          }
case 1:{
result = 5;
break;
          }
case 2:{
result = 8;
break;
          }
       }
    }
return result;
}
```

现在，我们要求 Table View 首先呈现 3 个 Section，第一个 Section3 行；第二个 Section5 行，第三个 Section8 行。然后呢？返回 Table View cell 的静态实例给 tableview，我们想要 tablview 呈现的 cells 如下：

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

    UITableViewCell *result = nil;

if ([tableView isEqual:self.myTableView]){

static NSString *TableViewCellIdentifier = @"MyCells";

result = [tableView
dequeueReusableCellWithIdentifier:TableViewCellIdentifier];

if (result == nil){
result = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:TableViewCellIdentifier];
       }

result.textLabel.text = [NSString stringWithFormat:@"Section %ld, Cell %ld",
                                        (long)indexPath.section,
                                        (long)indexPath.row];
```

```
    }

return result;

}
```

现在如果我们在模拟器中运行程序，我们将看到我们的成果:

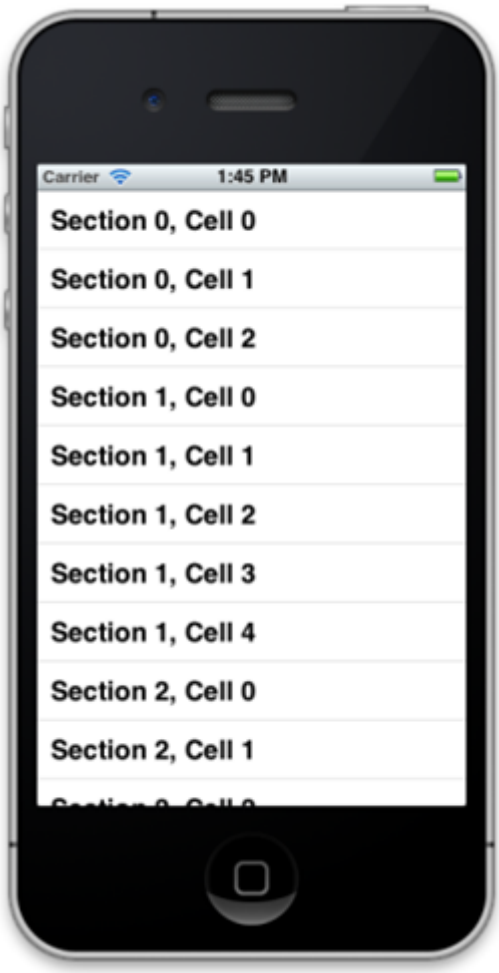

图 3-2. 一个包含 3 个 Section 的空白 Table View

当重新加载或刷新 Table View,要通过 UITableViewDataSource 协议查询数据源，请求许多的信息。在前面提到的重要方法中，Table View 首先请求 Section 的个数，每个 Section 来负责持有行或 cells。数据源确认了 Section 数目之后，Table View 会为每个 section 请求行数，数据源获取每个 Section 的从零开始的索引，并基于这一点，可以决定多少 cells 已加载到每个 Section。

确定了每个 Section 的 cells 数目之后，Table View 继续请求数据源以展示每个 Section 的每个 cells，你可以分配 UITableViewCell 类的实例并把其返回到 Table View。当然，每个 cells 可以设置的属性有很多，属性中包括标题、子标题，cels 的颜色以及更多。

3.3.4. 参考

XXX

3.4. 接收和处理 Table View 事件

3.4.1. 问题

你要响应 Table View 产生的各种事件。

3.4.2. 方案

给你的 Table View 提供一个委托对象。

这儿是一个带有 Table View 的视图控制器中.h 文件的代码片段：

```
#import <UIKit/UIKit.h>
@interface Receiving_and_Handling_Table_View_EventsViewController
                : UIViewController <UITableViewDelegate, UITableViewDataSource>
@property (nonatomic, strong) UITableView *myTableView;
@end
```

同一个视图控制器的.m 文中件实现 UITableViewDelegate 协议中定义的一个方法：

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

if ([tableView isEqual:self.myTableView]){

NSLog(@"%@",
      [NSString stringWithFormat:@"Cell %ld in Section %ld is selected",
       (long)indexPath.row, (long)indexPath.section]);
   }

}
- (void)viewDidLoad {
   [super viewDidLoad];

   self.myTableView = [[UITableView alloc]
initWithFrame:self.view.bounds
style:UITableViewStylePlain];

   self.myTableView.autoresizingMask =
      UIViewAutoresizingFlexibleHeight |
UIViewAutoresizingFlexibleWidth;
   self.myTableView.dataSource = self;
   self.myTableView.delegate = self;

   [self.view addSubview:self.myTableView];

}
- (void)viewDidUnload{
   [super viewDidUnload];
   self.myTableView = nil;
}
```

3.4.3. 讨论

当一个数据源负责给 Table View 提供数据时，无论何时事件发生 Table View 都将咨询委托，或者在完成一个任务之前如果 Table View 请求更多信息，它要调用一个委托的方法：

- 当一个 cell 被选中或者选中之前
- 当一个 Table View 需要找出每个 cell 的高度时
- 当一个 Table View 需要构造每个 section 表头和页脚时

正如你在本秘诀方案中看到的样本代码一样，当前对象被设置成一个 Table View 的委托；当用户选择一个 Table View 的一个单元格或者一行时，为了获得通知这个委托执行 tableView:didSelectRowAtIndexPath: selector。SDK 中 UITableViewDelegate 协议的文档向你展示了一个委托可以定义的、视图可以调用的所有方法。

3.4.4. 参考

"UITableViewDelegate Protocol Reference,"
http://developer.apple.com/library/ios/#documentation/uikit/reference/UITableViewDelegate_Protocol/Reference/Reference.html

3.5. 在 Table View 中使用不同种类的附件

3.5.1. 问题

通过展示不同附件，你想抓住用户对一个 Table View 的关注，或者提供不同的方式使用户在你的 Table View 中和每个 cells 互动。

3.5.2. 方案

使用 UITableViewCell 类中的 accessoryType 供给 Table View 数据源的实例对象：

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

    UITableViewCell* result = nil;

if ([tableView isEqual:self.myTableView]){

static NSString *MyCellIdentifier = @"SimpleCell";

        /* We will try to retrieve an existing cell
with the given identifier */
result = [tableView
dequeueReusableCellWithIdentifier:MyCellIdentifier];

if (result == nil){
          /* If a cell with the given identifier does not
exist, we will create the cell with the identifier
and hand it to the table view */

result = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:MyCellIdentifier];
        }

        result.textLabel.text =
        [NSString stringWithFormat:@"Section %ld, Cell %ld",
          (long)indexPath.section,
          (long)indexPath.row];

        result.accessoryType = UITableViewCellAccessoryDetailDisclosureButton;
```

```
    }

return result;

}
- (NSInteger) tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
return 10;
}
- (void)viewDidLoad{
    [super viewDidLoad];

self.myTableView = [[UITableView alloc] initWithFrame:self.view.bounds
style:UITableViewStylePlain];

    self.myTableView.dataSource = self;
    self.myTableView.delegate = self;

    self.myTableView.autoresizingMask =
        UIViewAutoresizingFlexibleWidth |
UIViewAutoresizingFlexibleHeight;
    [self.view addSubview:self.myTableView];

}
- (void)viewDidUnload{
    [super viewDidUnload];
    self.myTableView = nil;
}
```

3.5.3. 讨论

你可以把 UITableViewCellAccessoryType 枚举中定义的任何值分配到一个 UITableViewCell 类的实例中 accessoryType 的属性。*disclosure indicator* 和 *detail disclosure* 按钮是两个非常有用的附件。它们都通过袖章形箭头向用户指示，如果他们在关联的 Table View 单元格被点击，将显示新的视图或视图控制器。换句话说，用户将看到一个有更多关于当前所选项信息的屏幕。这两个附件的区别在于 disclosure indicator 不产生事件，而 detail disclosure 按钮在被点击时会向委托触发一个事件；也就是说，点击 cell 上的按键会有不同效果。因此，detail disclosure 按钮允许用户在同一行上执行两个独立但相关的操作。

图 3-3 显示在一个 Table View 中展示了这两个不同附件，第一个有 disclosure indicator，第二个有 detail disclosure 按钮。

图 3-3. 2 个有不同附件的 Table Viewcelss

如果你点击任何一个分配到 Table Viewcell 中的 detail disclosure 按钮，你会马上意识到它事实上是一个独立按钮。现在问题来了：当用户点击这个按钮的时候 Table View 如何知道呢？

就像之前解释的那样，Table View 在其委托对象上产生事件，一个 Table Viewcell 的 detail dis- closure 按钮也会产生一个被 Table View 的委托对象捕获的事件：

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

   /* Do something when the accessory button is tapped */
NSLog(@"Accessory button is tapped for cell at index path = %@", indexPath);

   UITableViewCell *ownerCell = [tableView cellForRowAtIndexPath:indexPath];

NSLog(@"Cell Title = %@", ownerCell.textLabel.text);

}
```

此代码查找哪个 table view cell 的 detail disclosure 按钮被触发，并将该 cell 的文本标签的内容打印到控制台屏幕。作为提醒，可以通过选择 Xcode 中的 Run→Console 来显示控制台屏幕。

3.5.4. 参考

XXX

3.6. 创建自定义 Table View 单元格附件

3.6.1. 问题

iOS SDK 提供给你的附件不满意，你想要创建自己的附件。

3.6.2. 方案

把类型 UIView 的一个实例分配到任何 UITableViewCell 类的实例的 accessoryView 的属性：

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

    UITableViewCell* result = nil;

static NSString *MyCellIdentifier = @"SimpleCell";

    /* We will try to retrieve an existing cell
with the given identifier */
result = [tableView dequeueReusableCellWithIdentifier:MyCellIdentifier];

if (result == nil){
result = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:MyCellIdentifier];
    }

result.textLabel.text = [NSString stringWithFormat:@"Section %ld, Cell %ld",
                                        (long)indexPath.section,
                                        (long)indexPath.row];
    UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
button.frame = CGRectMake(0.0f, 0.0f, 150.0f, 25.0f);

    [button setTitle:@"Expand"
forState:UIControlStateNormal];

    [button addTarget:self
action:@selector(performExpand:)
forControlEvents:UIControlEventTouchUpInside];

    result.accessoryView = button;

return result;

}
```

如你所见，这段代码使用 performExpand:方法作为每个按键的选择器。此处是这个方法的定义：

```
- (void) performExpand:(id)paramSender{
    /* Take an action here */
}
```

此示例代码段将自定义按钮分配到目标表中的每一行的辅助视图。结果如图 3-4 所示：

图 3-4. 带有自定义附件视图的 Table View 单元格

3.6.3. 讨论

UITableViewCell 类型的对象保留一个名为 accessoryView 属性；这是可以赋值的视图，如果你对内置 iOS SDK Table Viewcell 的附件不完全满意的话。属性设置之后，Cocoa Touch 将忽略 accessoryType 属性的值，同时把分配到 accessoryView 属性的值作为分配的附件用到单元格中，

这个方案中罗列的代码用于创建填充到 Table View 中的所有 cell 按钮。当点击任何单元的按钮，performExpand: 方法就会被调用；如果你像我一样的话，你可能已经开始思考关于你能如何决定发送按钮属于哪一个单元格的问题了。所以，现在我们要以某种方式把我们的按钮与其所属的 cell 连接起来。

处理这个问题的一个办法是检索触发事件的按钮的 superview。Table Viewcell 的辅助视图作为他们的子视图添加到 cell 的配件视图，因为检索按钮的 superview 将返回 Table Viewcell，返回的这个拥有配套的视图按钮的 cell：

```
- (void) performExpand:(UIButton *)paramSender{

    UITableViewCell *ownerCell = (UITableViewCell*)paramSender.superview;

if (ownerCell != nil){

        /* Now we will retrieve the index path of the cell
which contains the section and the row of the cell */

        NSIndexPath *ownerCellIndexPath =
          [self.myTableView indexPathForCell:ownerCell];

NSLog(@"Accessory in index path is tapped. Index path = %@",
ownerCellIndexPath);

        /* Now we can use these two values to truly determine that
the accessory button of which cell was the sender of this event:

          OwnerCellIndexPath.section
          OwnerCellIndexPath.row
```

```
        */
if (ownerCellIndexPath.section == 0 &&
ownerCellIndexPath.row == 1){
          /* This is the second row in the first section */
       }

       /* And so forth with the other checks ... */

    }

}
```

3.6.4. 参考

XXX

3.7. 在 Table View 中展示分层数据

3.7.1. 问题

你想能在一个 Table View 中显示分层数据。

3.7.2. 方案

使用 Table View cells 的缩进功能：

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

    UITableViewCell* result = nil;

static NSString *MyCellIdentifier = @"SimpleCells";

result = [tableView dequeueReusableCellWithIdentifier:MyCellIdentifier];

if (result == nil){
result = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:MyCellIdentifier];
    }

result.textLabel.text = [NSString stringWithFormat:@"Section %ld, Cell %ld",
                                    (long)indexPath.section,
                                    (long)indexPath.row];

    result.indentationLevel = indexPath.row;
    result.indentationWidth = 10.0f;

return result;

}
```

为了给每个 cell 的内容视图一个边距，用缩进等级与缩进宽度简单相乘。图 3-5 描述了这些 cell 在 Table View 展示时的外观：

图 3-5. 带缩进的 Table View 单元格

3.7.3. 讨论

虽然可能很少发现很有用，但是你可以在 iOS SDK 的 Table View cell 应用缩进。每个 cell 有 2 个相关属性：缩进等级和缩进宽度。缩进等级与缩进宽度简单相乘，所得的结果就是偏移量，根据这个值 Table Viewcell 的内容会向左右两侧偏移。

例如，如果一个 cell 的缩进等级设置为 2，缩进宽度设置为为 3，那么相乘的结果为 6；这就意味着当呈现在 Table View 中时 cell 内容视图向右移动 6 个像素。

缩进级别定义为有符号的整数值，使你可以将分配给它的值为负．这显然将使单元格的内容视图向左偏移。Table Viewcell 的缩进级别允许程序员呈现分层数据，并由程序员来确定每个 cell 的缩进级别和缩进宽度。

3.7.4. 参考

XXX

3.8. 启用 Table Viewcell 的滑动删除

3.8.1. 问题

想让你的 APP 用户能从 Table View 中轻松删除行。

3.8.2. 方案

在 Table View 的委托中执行 tableView:editingStyleForRowAtIndexPath: 选择器，在数据源执行 tableView:commitEditingStyle:forRowAtIndexPath:选择器：

```
- (UITableViewCellEditingStyle)tableView:(UITableView *)tableView
editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath{

    UITableViewCellEditingStyle result = UITableViewCellEditingStyleNone;

if ([tableView isEqual:self.myTableView]){
result = UITableViewCellEditingStyleDelete;
    }

return result;

}
- (void) setEditing:(BOOL)editing
animated:(BOOL)animated{

    [super setEditing:editing
animated:animated];

    [self.myTableView setEditing:editing
animated:animated];


}
- (void)    tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath{

if (editingStyle == UITableViewCellEditingStyleDelete){

if (indexPath.row < [self.arrayOfRows count]){

        /* First remove this object from the source */
        [self.arrayOfRows removeObjectAtIndex:indexPath.row];

        /* Then remove the associated cell from the Table View */
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
withRowAnimation:UITableViewRowAnimationLeft];

    }
  }

}
```

tableView:editingStyleForRowAtIndexPath:方法能够启动删除功能，它被 tabl view 调用，同时它的返回值决定了 table view 允许用户做什么(插入、删除等.)。

tableView:commitEditingStyle:forRowAt

IndexPath:方法实现用户要求的删除。后一种方法在委托中定义，但是它的功能有点重载：不只使用这方法删除数据，也必须要从表中删除行。

3.8.3. 讨论

Table View 通过在目标行右侧显示一个按钮来响应滑动（图 3-6）。正如你所看到的，Table View 不在编辑模式下，但按钮允许用户删除行。通过执行 tableView:editingStyleForRowAtIndexPath: (在 UITableViewDelegate 声明了)方法能够启动这个模式，它的返回值说明了表中是否应该允许同时插入与删除或者同时不。通过实现 Table View 的数据源中的 tableView:commitEditingStyle:forRowAtIndexPath:方法，你将会得到通知如果用户完成了插入或者删除操作的话。

图 3-6. 删除键出现在 Table View 单元格中

deleteRowsAtIndexPaths:withRowAnimation:方法的第二个参数允许你指定一个动画方法，当行从 Table View 中删除时这个动画方法会被执行。我们的示例说明了当行被删除时它在从右到左的移动过程中消失。

3.8.4. 参考

XXX

3.9. 在 Table View 中构建页眉和页脚

3.9.1. 问题

要在一个 Table View 中创建一个页眉和/或页脚

3.9.2. 方案

创建一个视图（可以是一个标签、图片等，任何可以直接或者间接的视图子类），然后把这个视图分配到一个 Table View 的 1 个 Section 的页眉和/或页脚。你也可以将特定数目的点作为高度分配到页眉或页脚，我们很快就会看到。

3.9.3. 讨论

Table View 可以有多个页眉页脚。一个 Table View 的每个 Section 都可有有它自己的页眉页脚，如果在一个 table view 中有 3 个 Section，你就最多有 3 个页眉和 3 个页脚。你不必给这些 Section 的每个部分添加页眉页脚。告诉 Table View 你是否在一个 Section 要页眉或页脚，通过其委托你是否把这些视图传递到 Table View 中，你想在 Table View 中为 Section（多个）提供页眉（多的）、页脚（多个），这些都取决于你。Table view 中的页眉页脚成了 Table View 的一部分，这就意味着当 Table View 的内容滚动时，Table View 中的页眉页脚也同样会滚动。我们看下一个 Table View 中的示例页眉页脚：

图 3-7. Table View 中顶部的页脚和最后一 Section 的页眉快捷方式

你可以看到，顶部的项目，如"拼写检查"和"启用 Caps Lock"有一个脚注，说"双击空格

键将插入一个时期后跟一个空格."。这是 Table View 的顶部区域的页脚。它是页脚而不是页眉的理由是它连接到该部分的底部，而不是其顶部。Table View 中的最后一部分还有一个标题，内容是"快捷方式"。它之所以是页眉而不是页脚，是由于出现在此 Section 的顶部而不是底部。通过 UITableViewDataSource 中定义的方法可以确定一个 Section 内页眉和页脚的高度。

通过 UITableViewDelegate 协议中定义的方法可以确定为一个表视图中一个 Section 的页眉/页脚展示的实际视图。

我们来创建一个带有一个 Table View 的简单 APP。随后我们提供 2 个标签，它们属于 UILable 类；一个做页眉，另一个作为表框内一个仅有 Section 的页脚；我们仅用 3 个 cell 填充这个 table view。在页眉中我们放入文字“Section 1Header”，在页脚标签上放入文字“Section1Footer”；以根视图控制器中的文件夹开始，我们定义一个 table view：

```
#import <UIKit/UIKit.h>
@interface Constructing_Headers_and_Footers_in_Table_ViewsViewController
                : UIViewController <UITableViewDataSource, UITableViewDelegate>
@property (nonatomic, strong) UITableView *myTableView;
@end
```

然后我们将要在视图控制器的实现文件中合成此属性：

```
#import "Constructing_Headers_and_Footers_in_Table_ViewsViewController.h"
@implementation Constructing_Headers_and_Footers_in_Table_ViewsViewController
@synthesize myTableView;
...
```

现在我们将创建一个分组的表视图，并给它加载三个单元格：

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

   UITableViewCell *result = nil;

static NSString *CellIdentifier = @"CellIdentifier";

result = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

if (result == nil){
result = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier];
   }

result.textLabel.text = [[NSString alloc] initWithFormat:@"Cell %ld",
                                    (long)indexPath.row];

return result;
}
- (NSInteger) tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
return 3;
}
- (void)viewDidLoad{
```

```
    [super viewDidLoad];

    self.myTableView =
    [[UITableView alloc] initWithFrame:self.view.bounds
style:UITableViewStyleGrouped];

    self.myTableView.dataSource = self;
    self.myTableView.delegate = self;
    self.myTableView.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                                        UIViewAutoresizingFlexibleHeight;

    [self.view addSubview:self.myTableView];

}
- (void)viewDidUnload{
    [super viewDidUnload];
    self.myTableView = nil;
}
- (BOOL)shouldAutorotateToInterfaceOrientation
            :(UIInterfaceOrientation)interfaceOrientation{
return YES;
}
```

这个部分令人振奋。现在我们可以使用 UITableViewDelegate 定义的 2 个重要方法提供一个页眉标签和另一个页脚标签，这个页脚标签属于我们已经加载到 Table View 中仅有的一个 Section。这 2 个方法分别是：

tableView:viewForHeaderInSection:

此方法预计一个 UIView 类型的返回值。此方法返回的视图将显示为 viewForHeaderInSection 参数指定部分的页眉。

tableView:viewForFooterInSection:

此方法预计一个 UIView 类型的返回值。此方法返回的视图将显示为 viewForFooterInSection 参数指定部分的页脚。

现在我们的任务是执行这些方法，并从其中返回一个 UILabel 的实例。按照计划，在页眉标签我们放入"Section 1 Header"，在页脚标签放入文字"Section 1 Footer"。

```
- (UIView *)    tableView:(UITableView *)tableView
viewForHeaderInSection:(NSInteger)section{

    UILabel *result = nil;

if ([tableView isEqual:self.myTableView] &&
section == 0){
result = [[UILabel alloc] initWithFrame:CGRectZero];
        result.text = @"Section 1 Header";
        result.backgroundColor = [UIColor clearColor];
        [result sizeToFit];
    }

return result;

}
- (UIView *)    tableView:(UITableView *)tableView
viewForFooterInSection:(NSInteger)section{

    UILabel *result = nil;
```

```
if ([tableView isEqual:self.myTableView] &&
section == 0){
result = [[UILabel alloc] initWithFrame:CGRectZero];
        result.text = @"Section 1 Footer";
        result.backgroundColor = [UIColor clearColor];
        [result sizeToFit];
    }

return result;

}
```

如果你现在在 iPhone 模拟器上运行你的 APP，你肯定会看到一些奇怪的事情，如图 3-8 所示：

Figure 3-8. 表框中没有正确对齐的页眉和页脚标签

出现这种标签偏差是因为 Table View 事实上不知道这些视图的高度。为了确定页眉和页脚视图的高度，我们需要使用下面 2 个在 UITableViewDelegate 协议中定义的方法：

tableView:heightForHeaderInSection:

这个方法的返回值是 CGFloat 类型，它确定了 Table View 中一个 Section 的页眉高度，这个 Section 的索引通过 heightForHeaderInSection 参数传递。

tableView:heightForFooterInSection:

这个方法的返回值是 CGFloat 类型，它确定 Table View 中一个 Section 的页脚高度，这个 Section 的索引通过 heightForHeaderInSection 参数传递。

```
- (CGFloat) tableView:(UITableView *)tableView
heightForHeaderInSection:(NSInteger)section{

   CGFloat result = 0.0f;
```

```
if ([tableView isEqual:self.myTableView] &&
section == 0){
result = 30.0f;
    }

return result;

}
- (CGFloat) tableView:(UITableView *)tableView
heightForFooterInSection:(NSInteger)section{

    CGFloat result = 0.0f;

if ([tableView isEqual:self.myTableView] &&
section == 0){
result = 30.0f;
    }

return result;

}
```

运行这个 App，你会看到页眉和页脚标签的高度是固定的。我们写的代码中仍然有错误，问题在页眉和页脚标签的左边距上。自己看一下（见图 3-9）：

图 3-9. 页眉页脚标签的左边距是错误的。

这背后的原因是在默认情况下，表视图 Table View 把页眉和页脚视图放置在 x 点 0.0f。你可能想到通过更改页眉标签的 frame 可以解决这个问题，但不幸的是这个方法行不通。这个问题的方案是创建一个通用的 UIView，并把页眉页脚标签放到上面。返回作为页眉/页脚的通用视图，但是要在通用视图内更改标签的 x 位置。现在我们需要修正

tableView:viewForHeaderInSection: 和 tableView:viewForFooterInSection:方法的实现过程：

```
- (UIView *)    tableView:(UITableView *)tableView
viewForHeaderInSection:(NSInteger)section{
   UIView *result = nil;

if ([tableView isEqual:self.myTableView] &&
section == 0){

      UILabel *label = [[UILabel alloc] initWithFrame:CGRectZero];
      label.text = @"Section 1 Header";
      label.backgroundColor = [UIColor clearColor];
      [label sizeToFit];

      /* Move the label 10 points to the right */
label.frame = CGRectMake(label.frame.origin.x + 10.0f,
                                    5.0f, /* Go 5 points down in y axis */
                                    label.frame.size.width,
                                    label.frame.size.height);

      /* Give the container view 10 points more in width than our label
because the label needs a 10 extra points left-margin */
      CGRect resultFrame = CGRectMake(0.0f,
                                          0.0f,
                                          label.frame.size.height,
                                          label.frame.size.width + 10.0f);
result = [[UIView alloc] initWithFrame:resultFrame];
      [result addSubview:label];

   }

return result;

}
- (UIView *)    tableView:(UITableView *)tableView
viewForFooterInSection:(NSInteger)section{

   UIView *result = nil;

if ([tableView isEqual:self.myTableView] &&
section == 0){

      UILabel *label = [[UILabel alloc] initWithFrame:CGRectZero];
      label.text = @"Section 1 Footer";
      label.backgroundColor = [UIColor clearColor];
      [label sizeToFit];

      /* Move the label 10 points to the right */
label.frame = CGRectMake(label.frame.origin.x + 10.0f,
                                    5.0f, /* Go 5 points down in y axis */
                                    label.frame.size.width,
                                    label.frame.size.height);

      /* Give the container view 10 points more in width than our label
because the label needs a 10 extra points left-margin */
      CGRect resultFrame = CGRectMake(0.0f,
                                          0.0f,
                                          label.frame.size.height,
                                          label.frame.size.width + 10.0f);
```

```
result = [[UIView alloc] initWithFrame:resultFrame];
    [result addSubview:label];

  }

return result;

}
```

现在你运行 APP 的话将看到类似于图 3-10 中显示的结果：

图 3-10. Table View 中展示出来的页眉页脚标签

根据你刚了解到的这些方法，你甚至可以在 Table View 中放置图片做页眉页脚。UIImageView 的实例把 UIView 作为它们的父类，所以你可以轻松地把你的图片放进图片视图中，并将它们作为页眉页脚返回。如果你想在 Table View 中放入文字作为页眉/页脚，你可以利用 UITableViewDataSource 协议中定义的 2 个简便方法，这些方法能为你省去不少麻烦和时间。不需要创建你自己的标签，并将它们返回作为 Table View 的页眉/页脚，可以简单地使用这些方法：

tableView:titleForHeaderInSection:

这个方法的返回值是 NSString 类型，字符串被 Table View 自动设置到 UILabel，将会在 Section 的页眉部分展示出来，这个 Section 由 titleForHeaderInSection 参数确定。

tableView:titleForFooterInSection:

这个方法的返回值是 NSString 类型，字符串被 Table View 自动设置到 UILabel，将会在 Section 的页脚部分展示出来，这个 Section 由 titleForFooterInSection 参数确定。

所以为了使我们的 App 代码更简单，让我们摆脱执行 tableView:viewForHeaderInSection: 和 tableView:viewForFooterInSection: 方法，并且通过执行 tableView:titleForHeaderInSection: 和 tableView:titleForFooterInSection:方法来代替它们：

```
- (NSString *) tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section{

    NSString *result = nil;

if ([tableView isEqual:self.myTableView] &&
section == 0){
result = @"Section 1 Header";
    }

return result;

}
- (NSString *) tableView:(UITableView *)tableView
titleForFooterInSection:(NSInteger)section{

    NSString *result = nil;

if ([tableView isEqual:self.myTableView] &&
section == 0){
result = @"Section 1 Footer";
    }

return result;

}
```

现在在 iPhone 模拟器上运行 App，你会看到 Table View 为其唯一一个 Section 的页眉和页脚分别自动创建了一个左对齐标签和一个中对齐 UILabel；这些 UILabel 对齐方式是默认对齐，也就是每个视图用这些 UILabel 创建了自己的页眉页脚。

(见图 3-11):

图 3-11. Table View 在页眉和页脚中呈现文本

3.9.4. 参考

XXX

3.10. 在 Table Viewscell 中展示快捷菜单

3.10.1. 问题

你想让用户使用 App 时，只要通过一个手指放在 App 中一个 Table Viewcell 上，就能在他们原本可选的操作中使用复制/粘贴选项。

3.10.2. 方案

在你的 Table View 的委托对象上实现下面 3 个 UITableViewDelegate 协议的方法：

tableView:shouldShowMenuForRowAtIndexPath:

这个方法的返回值属于 BOOL 类型。如果你从这个方法返回 Yes，iOS 将为 Table Viewcell 展示快捷菜单，这个 cell 的索引通过 shouldShowMenuForRowAtIndexPath 参数传递给你。

tableView:canPerformAction:forRowAtIndexPath:withSender:

这个方法的返回值也属于 BOOL 类型。一旦你允许 iOS 为 Table Viewcell 展示快捷菜单，iOS 将会多次调用这个方法，并且通过你对选择器的操作来决定是不是要展示快捷菜单。所以如果 iOS 会问你是否想要对用户显示“复制”菜单，这个方法就会在 Table View 的委托对象中调用；这个方法的 canPerformAction 参数等同于@selector(copy:). 我们将在下文对这个方法*讨论*部分得到更多信息。

tableView:performAction:forRowAtIndexPath:withSender:

在 Table Viewcell 的快捷菜单中一旦你允许显示特定动作，并且一旦用户从菜单中选中了这个动作时，这个方法会在 Table View 的委托对象中调用。此时，你必须采取任何满足用户需求的行动。例如，如果用户选择的“Copy”，你将需要有一个粘贴板放那些被选中的 Table View 单元格的内容。

3.10.3. 讨论

Table View 会给 iOS 一个是/否的回答，允许或者不允许它为 Table Viewcell 显示可用系统菜单选项。当用户把手指放在 cell 一个确定时间内，严格来说大约 1 秒钟，iOS 试图在 Table View cell 上显示一个快捷菜单。随后 iOS 会提问 Table View 其 cell 是菜单触发的来源。如果 Table View 给了 iOS 一个肯定的回答，那么 iOS 随后会告诉 Table View 什么选项适合显示于快捷菜单，Table View 才能够对任一选项说是或者否。如果有 5 个适用于实例的菜单选项，Table View 只对其中的 2 个说是；那么只有 2 个选项显示出来。在菜单选项显示给用户之后，用户既能点击其中一个项目或者点击快捷菜单的外围来取消点击效果。一旦用户点击了其中一个菜单选项，iOS 将发送一条委托消息到 Table View 通知它用户点击的菜单选项。基于所有这些信息，Table View 可以做一个决定，要对这些已经选择的动作做什么。我建议我们先看看执行什么动作适用于 Table Viewcell 的快捷菜单，所以我们先创建一个 Table View 然后在其中展示一些 cell：

```
- (NSInteger) tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
return 3;
}
- (UITableViewCell *) tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath{
```

```
    UITableViewCell *result = nil;

static NSString *CellIdentifier = @"CellIdentifier";

result = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];

if (result == nil){
result = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier];
    }

    result.textLabel.text = [[NSString alloc]
initWithFormat:@"Section %ld Cell %ld",
                                        (long)indexPath.section,
                                        (long)indexPath.row];

return result;

}
- (void)viewDidLoad{
    [super viewDidLoad];

    self.view.backgroundColor = [UIColor whiteColor];

    self.myTableView = [[UITableView alloc]
initWithFrame:self.view.bounds
style:UITableViewStylePlain];

    self.myTableView.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                                                UIViewAutoresizingFlexibleHeight;

    self.myTableView.dataSource = self;
    self.myTableView.delegate = self;

    [self.view addSubview:self.myTableView];

}
- (void)viewDidUnload{
    [super viewDidUnload];
    self.myTableView = nil;
}
```

现在我们将执行前面提到的在 UITableViewDelegate 协议中定义的 3 个方法，把可使用的动作（SEL 类型）简单转化成字符串并且打印到控制台：

```
- (BOOL)                        tableView:(UITableView *)tableView
shouldShowMenuForRowAtIndexPath:(NSIndexPath *)indexPath{

    /* Allow the context menu to be displayed on every cell */
return YES;

}
- (BOOL) tableView:(UITableView *)tableView
canPerformAction:(SEL)action
forRowAtIndexPath:(NSIndexPath *)indexPath
withSender:(id)sender{

NSLog(@"%@", NSStringFromSelector(action));
```

```
   /* Allow every action for now */
return YES;
}
- (void) tableView:(UITableView *)tableView
performAction:(SEL)action
forRowAtIndexPath:(NSIndexPath *)indexPath
withSender:(id)sender{

   /* Empty for now */

}
```

现在在模拟器或者设备上运行 APP，你将看到 3 个 cell 加载到了 Table View。用手指（如果使用设备）或者指针（如果适用 iOS 模拟器）点击其中一个 cell，观察控制台窗口会打印出什么：

```
cut:
copy:
select:
selectAll:
paste:
delete:
_promptForReplace:
_showTextStyleOptions:
_define:
_accessibilitySpeak:
_accessibilityPauseSpeaking:
makeTextWritingDirectionRightToLeft:
makeTextWritingDirectionLeftToRight:
cut:
copy:
select:
selectAll:
paste:
delete:
_promptForReplace:
_showTextStyleOptions:
_define:
_accessibilitySpeak:
_accessibilityPauseSpeaking:
makeTextWritingDirectionRightToLeft:
makeTextWritingDirectionLeftToRight:
```

你应该需要这些 iOS 允许你公开给用户的动作。例如，如果你仅仅想让你的用户有 Copy 选项，在 tableView:canPerformAction:forRowAtIndexPath:withSender:方法中简单找出那些在显示之前 iOS 向你提问要获得你的允许，同时返回 Yes 或者 No 的动作：

```
- (BOOL) tableView:(UITableView *)tableView
canPerformAction:(SEL)action
forRowAtIndexPath:(NSIndexPath *)indexPath
withSender:(id)sender{

if (action == @selector(copy:)){
return YES;
   }
```

```
return NO;
}
```

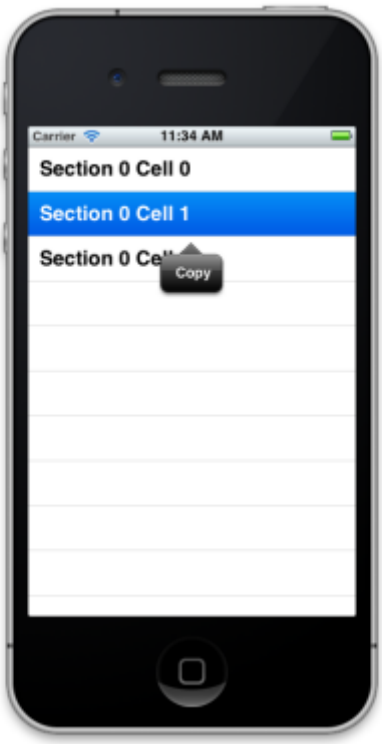

图 3-12. 在 Table View 单元格的下拉菜单中展示的 Copy 动作

下一步将捕获用户实际上从快捷菜单中选定的项目。在这些信息的基础上我们能选择合适的动作。例如，如果用户从快捷菜单中选择了 Copy，那为了后面使用我们就可以用 UIPasteBoard 把这些 cell 复制到粘贴板上：

```
- (void) tableView:(UITableView *)tableView
performAction:(SEL)action
forRowAtIndexPath:(NSIndexPath *)indexPath
withSender:(id)sender{

if (action == @selector(copy:)){
      UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];
      UIPasteboard *pasteBoard = [UIPasteboard generalPasteboard];
      [pasteBoard setString:cell.textLabel.text];

   }

}
```

3.10.4. 参考

XXX

3.11. 在 Table View 中移动 cell 和 Section

3.11.1. 问题

你想用流畅直观的动画来移动和拖曳 Table View 的 cell 和 Section。

3.11.2. 方案

用 Table View 的 moveSection:toSection:方法把一个 Section 移动到新位置。也可以使用 moveRowAtIndexPath:toIndexPath: 方法把一个 Table View cell 从当前位置移动到一个新位置。

3.11.3. 讨论

移动 Table View 的 cells 和 Section 和交换它们不一样，我们举个例子会更容易理解。假设你的 Table View 有 3 个 Section，SectionA, B 和 C。如果你把 Section A 移动到 Section C, Table View 会注意到这个移动，它会把 Section B 移动到 Section A 之前的位置，把 Section A 移动到 Section B 之前的位置。然而，如果 Section B 移动到了 Section C 的位置，Table View 没有必要移动 Section A，因为 Section A 在这个 Section 的顶端，不会影响到 Section B 和 Section C 的位置调整。在这个例子中，Section C 会移动到 Section B 的位置，SectionB 放到了 Section C 的位置。在移动 cell 时也用到同样的逻辑。为了说明这一点，我们可以创建一个 Table View 并在其中加载 3 个 Section，每个 Section 有 3 个 cell。我们从视图控制的页眉文档开始：

```
#import <UIKit/UIKit.h>
@interface Moving_Cells_and_Sections_in_Table_ViewsViewController
                : UIViewController <UITableViewDelegate, UITableViewDataSource>
@property (nonatomic, strong) UITableView *myTableView;
/* Each section is an array on its own, containing objects of type NSString */
@property (nonatomic, strong) NSMutableArray *arrayOfSections;
@end
```

我们的视图控制器会变成 Table View 的数据源。Table View 包含有 Section,每个 Section 中有行。所以我们要保持数组中的数组。我们的数组节的第一个数组，该数组本身将包含包含我们的 cell 和其它数组。在视图控制器的总的页眉文档中定义的 arrayOfSections 会完成这个任务，我们在视图控制器中实现该数组的填充：

```
#import "Moving_Cells_and_Sections_in_Table_ViewsViewController.h"
@implementation Moving_Cells_and_Sections_in_Table_ViewsViewController
@synthesize myTableView;
@synthesize arrayOfSections;
- (NSMutableArray *) newSectionWithIndex:(NSUInteger)paramIndex
withCellCount:(NSUInteger)paramCellCount{

   NSMutableArray *result = [[NSMutableArray alloc] init];

   NSUInteger counter = 0;
for (counter = 0;
counter< paramCellCount;
counter++){

      [result addObject:[[NSString alloc] initWithFormat:@"Section %lu Cell %lu",
                         (unsigned long)paramIndex,
```

```
                                (unsigned long)counter+1]];

    }

return result;

}
- (id) initWithNibName:(NSString *)nibNameOrNil
bundle:(NSBundle *)nibBundleOrNil{

self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil];

if (self != nil){

arrayOfSections = [[NSMutableArray alloc] init];

        NSMutableArray *section1 = [self newSectionWithIndex:1
withCellCount:3];
        NSMutableArray *section2 = [self newSectionWithIndex:2
withCellCount:3];
        NSMutableArray *section3 = [self newSectionWithIndex:3
withCellCount:3];
        [arrayOfSections addObject:section1];
        [arrayOfSections addObject:section2];
        [arrayOfSections addObject:section3];

    }

return self;

}
```

We shall then instantiate our table view and implement the necessary methods in the UITableViewDataSource protocol to populate our table view with data:

```
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView{

    NSInteger result = 0;

if ([tableView isEqual:self.myTableView]){
result = (NSInteger)[self.arrayOfSections count];
    }

return result;

}
- (NSInteger) tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{

    NSInteger result = 0;

if ([tableView isEqual:self.myTableView]){

if ([self.arrayOfSections count] > section){

          NSMutableArray *sectionArray = [self.arrayOfSections
objectAtIndex:section];
result = (NSInteger)[sectionArray count];

        }
```

```
    }

return result;

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

    UITableViewCell *result = nil;

if ([tableView isEqual:self.myTableView]){

static NSString *CellIdentifier = @"CellIdentifier";

result = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
if (result == nil){
result = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:CellIdentifier];
        }

        NSMutableArray *sectionArray = [self.arrayOfSections
objectAtIndex:indexPath.section];

result.textLabel.text = [sectionArray objectAtIndex:indexPath.row];

    }

return result;

}
- (void)viewDidLoad{
    [super viewDidLoad];

    self.myTableView =
    [[UITableView alloc] initWithFrame:self.view.bounds
style:UITableViewStyleGrouped];

    self.myTableView.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                                        UIViewAutoresizingFlexibleHeight;

    self.myTableView.delegate = self;
    self.myTableView.dataSource = self;

    [self.view addSubview:self.myTableView];

}
- (void)viewDidUnload{
    [super viewDidUnload];
    self.myTableView = nil;
}
```

时间显示！我们应该首先看下 Sections 是如何移动到新位置的吗,写一个能把 Section1 移动到 3 的方法：

```
- (void) moveSection1ToSection3{
```

```
    NSMutableArray *section1 = [self.arrayOfSections objectAtIndex:0];
    [self.arrayOfSections removeObject:section1];
    [self.arrayOfSections addObject:section1];

    [self.myTableView moveSection:0
toSection:2];

}
```

由于此刻我们的 UI 上没有按键，我要留给你自己决定调用这个方法的时间。你可以简单创建一个导航控制器，把控制的按键放在 UI 上，然后再调用这个方法。一旦你像以往那样运行 App，你将看到 Sections 从上到下按照 1 到 3 的次序排列：

图 3-13. Table View 包含 3 个 Sections，每个 Section 有 3 个单元格

在你调用了这个 moveSection1ToSection3 的方法之后，Section 1 会移动到 Section 3，Section 3 移到了 Section 2 以前的位置上，最后 Section 2 移到了 Section 1 以前的位置上：

图 3-14. Section 1 移到了 Section 3，其他 Sections 也随后移动了位置

移动 cell 和移动 Sections 很类似。要移动 cell，我们要做的就是使用 moveRowAtIndexPath:toIndexPath:方法。记住你能把一个 cell 在一个 Section 内移动或者移动到一个新 Section，简便起见，把 Section1 的 cell 1 移动到这个 Section 的 cell 2，看看将会发生什么：

```
- (void) moveCell1InSection1ToCell2InSection1{

   NSMutableArray *section1 = [self.arrayOfSections objectAtIndex:0];
   NSString *cell1InSection1 = [section1 objectAtIndex:0];
   [section1 removeObject:cell1InSection1];
   [section1 insertObject:cell1InSection1
atIndex:1];

   NSIndexPath *sourceIndexPath = [NSIndexPath indexPathForRow:0
inSection:0];
   NSIndexPath *destinationIndexPath = [NSIndexPath indexPathForRow:1
inSection:0];

   [self.myTableView moveRowAtIndexPath:sourceIndexPath
toIndexPath:destinationIndexPath];

}
```

代码将会发生什么呢？好，在我们移动了 cell 之后，我们需要确认数据源中那些要在 TableView 中显示的数据正确，所以我们要首先移动 Section 1 的 cell 1。把 cell 2 移动到 cell 1 和把 cell 3 移动到 cell 2，在一个数组中总共有 2 个 cell。然后我们要把 cell 1 插入数组的索引 1（第二个对象）。这样我们的数组包含 cell 1,2,3。这个做完之后，我们要在 Table View 中真正移动 cell 了。让这个问题更难一点，如何把 Section1 的 cell 2 移动到 Section2 的

第一个 cell？

```
- (void) moveCell2InSection1ToCell1InSection2{

    NSMutableArray *section1 = [self.arrayOfSections objectAtIndex:0];
    NSMutableArray *section2 = [self.arrayOfSections objectAtIndex:1];

    NSString *cell2InSection1 = [section1 objectAtIndex:1];
    [section1 removeObject:cell2InSection1];

    [section2 insertObject:cell2InSection1
atIndex:0];

    NSIndexPath *sourceIndexPath = [NSIndexPath indexPathForRow:1
inSection:0];
    NSIndexPath *destinationIndexPath = [NSIndexPath indexPathForRow:0
inSection:1];

    [self.myTableView moveRowAtIndexPath:sourceIndexPath
toIndexPath:destinationIndexPath];

}
```

这个转变的结果如图 3-15 所示：

图 3-15. 第一个 Section 的 cell 2 移动到了第二个 Section 的第 1 cell

3.11.4. 参考

XXX

3.12. 从 Table View 删除单元格和 Section

3.12.1. 问题

3.12.2. 方案

3.12.3. 讨论

3.12.4. 参考

XXX

点击这里访问：DevDiv.com 移动开发论坛