# 1 UITableView

UITableView可以在一栏内显示一系列的内容，它是UIScrollView的子类。但与UIScrollView可以任意方向滑动内容不同，UITableView只能在垂直方向上滑动来查看内容。整个UITableView由许多个UITableViewCell组成，UITableView中的内容也由UITableViewCell来负责显示。

在iOS应用程序中，UITableView是用于展示列表内容最常用的方法。UITableView在应用程序中可用于：a、展示具有层次结构的数据。b、展示可供选择的列表。c、对列表进行分组管理。d、展示带有索引的列表并根据索引快速找到所需要的数据。

## 1.UITableView和UITableViewCell的样式

在表格 tableview 初始化的时候我们可以指定需要使用的 UITableViewStyle 样式，可用的样式一共有两种：.plain 和 .grouped，默认下为.plain样式。

在plain模式下，如果UITableView有多个分区、分组，组与组之间是没有间隔的，并且组头组尾会有悬停效果，即列表滚动时组头组尾会自动悬停，而不是随着单元格一起移动。图1展示了在有组头时组头的悬停。

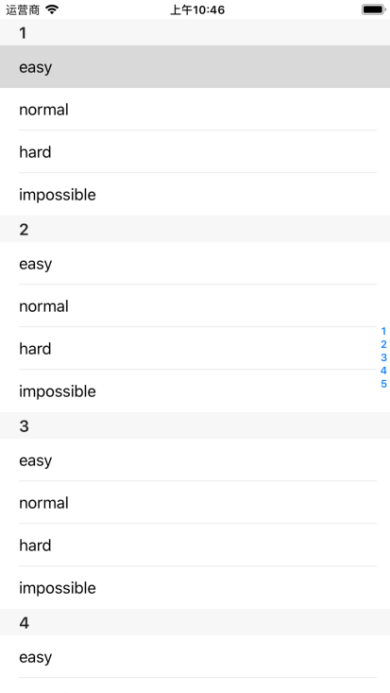
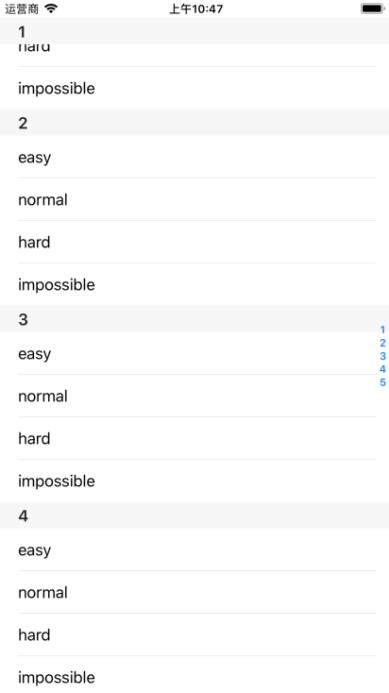
 

图1 UITableView的plain样式

在grouped模式下，如果UITableView有多个分区、分组，组与组之间是有间隔的，并且组头组尾没有悬停效果。

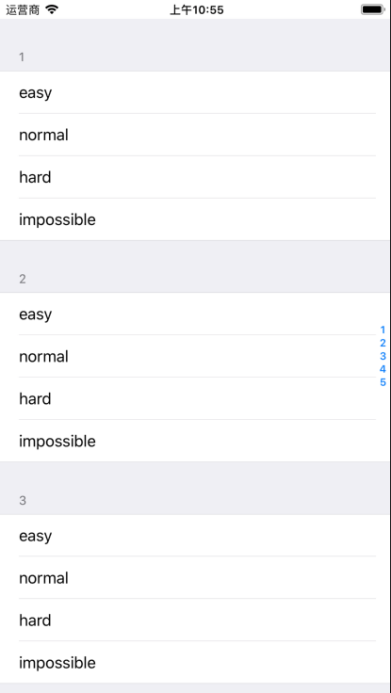
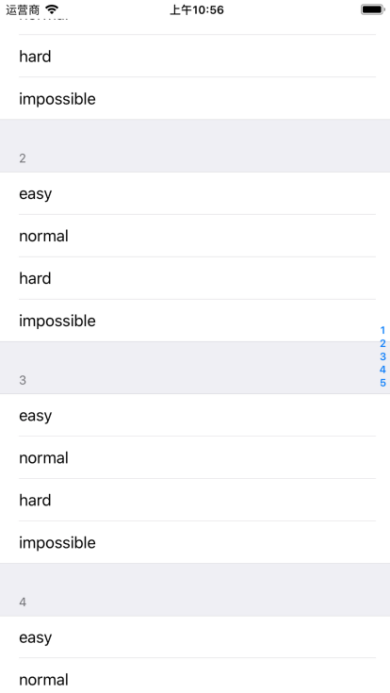
 

图2 UITableView的grouped样式

对于UITableViewCell，开发者可以自定义，也可以使用系统预定义的几种格式。系统预定义的格式有4种：.default、.value1、.value2、.subtitle，包含固定样式的标签、图标等对象，开发者只需对这些对象提供对应的文字和图像内容即可。使用textLable和detailTextLable属性设置文字内容，使用imageView属性设置单元格的图标。各样式显示形式如图3~6所示。

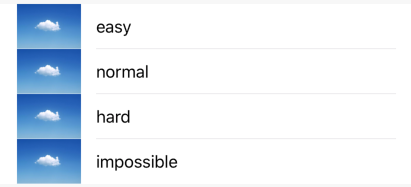
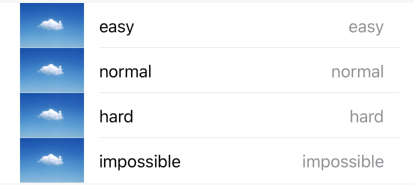
 

图3 default样式 图4 value1样式

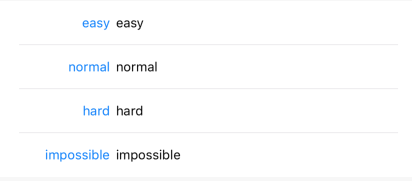
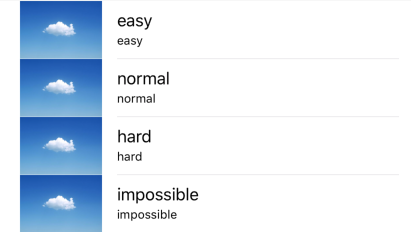
 

图5 value2样式 图6 subtitle样式

## 2.UITableView的简单使用

接下来会通过使用UITableView创建一个简单的表格。创建一个新项目，将ViewController.swift文件中的代码改写成如下代码。

1. import UIKit
2. //继承UITableViewDataSource
3. class ViewController: UIViewController,UITableViewDataSource {
4. override func viewDidLoad() {
5. super.viewDidLoad()
6. // Do any additional setup after loading the view, typically from a nib.
7. //设置UITableView大小
8. let tableRect = CGRect(x: 0, y: 20, width: UIScreen.main. bounds.width, height: UIScreen.main.bounds.height-20)
9. let tableView = UITableView(frame: tableRect)
10. //设置数据来源
11. tableView.dataSource=self
12. self.view.addSubview(tableView)
13. }
14. //设置UITableView表格行数
15. func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
16. return 20
17. }
18. //设置UITableViewCell单元格内容并使其能够复用
19. func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
20. let id="reused"
21. var cell = tableView.dequeueReusableCell(withIdentifier: id)
22. if(cell==nil){
23. cell=UITableViewCell(style: .default, reuseIdentifier: id)
24. }
25. cell?.textLabel?.text="这是表格内容"
26. return cell!
27. }
28. }

运行项目，结果如图7所示.



图7

在该代码中添加了UITableViewDataSource协议，用于设置UITableView表格中的数据来源，并处理数据源的变化。该协议可以实现的方法如表所示。

表1 UITableViewDataSource协议的代理方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| tableView(\_:cellForRowAt:) | 将指定位置的单元格初始化和复用，必须实现。 |
| tableView(\_:numberOfRowsInSection) | 设置某一章节单元格的数量，默认为1，必须实现。 |
| numberOfSections(in:) | 设置表格章节数，默认为1. |
| sectionIndexTitles(for:) | 设置表格索引内容。 |
| [tableView(\_:sectionForSectionIndexTitle:at:)](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614933-tableview) | 将表格索引和对应的表格标题连接起来。 |
| tableView(\_:titleForHeaderInSection:) | 设置指定章节的头部标题，若此方法返回值为nil，则该章节不显示标题。 |
| tableView(\_:titleForFooterInSection:) | 设置指定章节的尾部标题，若此方法返回值为nil，则该章节不显示尾部标题。 |
| tableVeiw(\_:canEditRowAt:) | 设置指定位置的单元格可被编辑，需返回值为true。 |
| [tableView(\_: commit:forRowAt:)](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614871-tableview) | 设置指定行的具体的插入或删除操作。 |
| [tableView(\_:canMoveRowAt:)](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614927-tableview) | 设置指定位置的单元格是否可以通过拖拽改变其位置。 |
| [tableView(\_:moveRowAt:to:)](https://developer.apple.com/documentation/uikit/uitableviewdatasource/1614927-tableview) | 当单元格从一个位置拖拽到另一个位置时，调用此方法。 |

UITableView有一个重要的属性：indexPath，其类型为NSIndexPath。NSIndexPath表示在嵌套数列的树中指定节点的路径，类似于磁盘中某文件夹的路径。UITableView使用indexPath来表示表格视图中指定章节的指定行。

indexPath包含两个元素：元素section表示表格章节序号，元素row表示章节中的行号。注意section和row的序号都是从0开始，如NSIndexPath(forRow:0,inSecton:0)表示UITableView中第一章节中的第一个单元格。

UITableView中的单元格cell利用复用机制来提高性能。假设整个屏幕能够显示10个单元格，当滑动表格时，当第一个单元格消失在屏幕时，该单元格就会放入复用池中，之后显示的单元格是从复用池中取出的单元格。这种机制大大降低了内存的损耗。试想，如果整个表格实际有100行，不用复用机制就需要创建100个单元格，使用复用机制只需要创建10个单元格。类似于酒店消耗筷子的数量，如果每双筷子用完即扔，那么酒店筷子的消耗量将十分庞大。但是如果筷子使用后经过洗涤和消毒，再提供给下一批客户使用，就大大降低了筷子的消耗。

在代码中，一般会定义一个字符串常量作为单元格cell的唯一标示，通过tableview的dequeueReusableCell方法从单元格对象池中获取指定标识，并且可重复使用的单元格cell。当单元格对象池中为空时，使用init(style: reuseIdentifier: )方法创建可复用单元格UITableViewCell对象并将其标识设为相应标识。具体操作可查看示例代码。对于相同类型的单元格，其reuseIdentifier参数必须相同。开发者可以对表格中的单元格设置不同的标识，以便对不同的单元格做不同的设置和操作。

## 3.表格的章节和索引

带有章节和和索引的表格在之前已经展示过了，索引位于表格右侧，点击索引序列会快速定位到对应的章节位置，这在表格有大量数据时很方便，现在的手机通讯录就是利用了这一点。

创建一个新项目，将ViewController.swift文件中的代码改写成如下代码。

1. import UIKit
2. class ViewController: UIViewController,UITableViewDataSource{
3. var indexString=["a","b","c","d","e"]
4. override func viewDidLoad() {
5. super.viewDidLoad()
6. // Do any additional setup after loading the view, typically from a nib.
7. let tableRect = CGRect(x: 0, y: 20, width: UIScreen.main.bounds.width, height: UIScreen.main.bounds.height-20)
8. let tableView = UITableView(frame: tableRect)
9. tableView.dataSource=self
10. self.view.addSubview(tableView)
11. }
12. //设置UITableView表格行数
13. func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
14. return 10
15. }
16. //设置UITableView表格章节数
17. func numberOfSections(in tableView: UITableView) -> Int {
18. return indexString.count
19. }
20. //设置索引内容
21. func sectionIndexTitles(for tableView: UITableView) -> [String]? {
22. return indexString
23. }
24. //设置每一章节头部标题
25. func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
26. return indexString[section]
27. }
28. //设置每一章节尾部标题
29. func tableView(\_ tableView: UITableView, titleForFooterInSection section: Int) -> String? {
30. return indexString[section]
31. }
32. //设置UITableViewCell单元格内容并使其能够复用
33. func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
34. let id="reused"
35. var cell = tableView.dequeueReusableCell(withIdentifier: id)
36. if(cell==nil){
37. cell=UITableViewCell(style: .default, reuseIdentifier: id)
38. }
39. cell?.textLabel?.text=indexString[indexPath.section]+String(indexPath.row)
40. return cell!
41. }
42. }

运行项目，点击表格右侧的索引，观察表格视图的跳转。可以看到，当点击索引‘c’时，表格直接跳转到‘c’章节，如图所示。

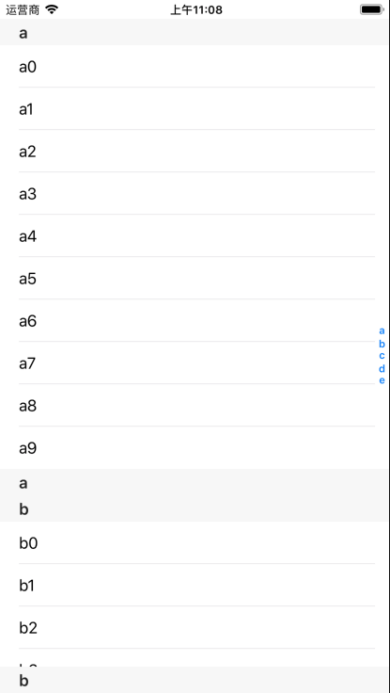
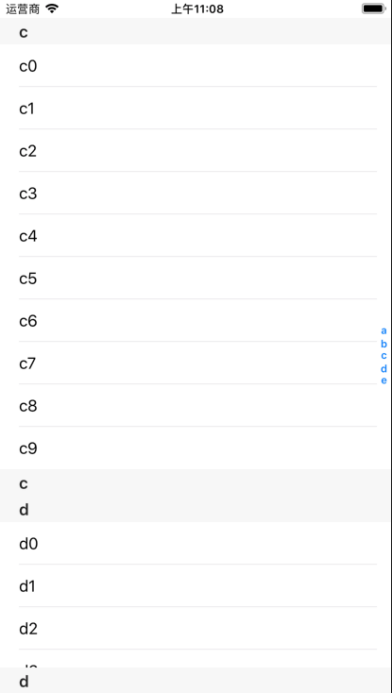
 

图8 点击索引跳转

## 4.自定义单元格

有时系统给的默认单元格格式无法满足需求，这就需要开发者自行定义单元格的格式。本节将介绍如何给不同的单元格设置不同的高度，以及设置点击单元格后的动作等。

首先需要介绍UITableViewDelegate协议，该协议通过UITableView的indexPath属性控制对指定单元格的选择、指定章节的头尾内容显示、单元格高度设置、单元格内容复制和粘贴等。UITableViewDelegate协议包含的方法很多，表2列出了其包含的几个常用代理方法,后面对单元格进行操作的方法都是包含在UITableViewDelegate协议中的。

表2 UITableViewDelegate协议的常用方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| tableView(\_:heightForRowAt:) | 设置指定索引位置单元格的高度。每当表格格要显示的时候都会调用此方法来设置表格中每个单元格的高度。 |
| tableView(\_:willDisplay:forRowAt:) | 当指定位置的单元格即将显示时调用此方法。 |
| tableView(\_:didSelectRowAt:) | 当点击到指定位置的单元格时，调用此方法。 |
| tableView(\_:didDeselectRowAt:) | 当点击到已经被选中的单元格时调用此方法 |
| tableView(\_:viewForHeaderInSection:) | 设置指定章节头部区域视图内容 |
| tableView(\_:viewForFooterInSection:) | 设置指定章节尾部区域视图内容 |
| tableView(\_:editingStyleForRowAt:) | 设置指定单元格的可编辑类型，如插入、删除 |

接下来修改第2节的项目代码，通过使用UITableViewDelegate协议的方法修改表格的格式。在第2节的代码第3行修改代码为：

1. class ViewController: UIViewController,UITableViewDataSource, UITableViewDelegate{

在第2节的代码第17行下方添加如下代码。

1. //修改各单元格高度
2. func tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
3. //将第二行单元格高度设为100区别开来
4. if(indexPath as NSIndexPath).row == 1{
5. return 100
6. }
7. return 40
8. }
9. //单元格被点击时输出相应行数
10. func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
11. print("第"+String(indexPath.row+1)+"行被点击")
12. }
13. //修改单元格背景颜色
14. func tableView(\_ tableView: UITableView, willDisplay cell: UITableViewCell, forRowAt indexPath: IndexPath) {
15. cell.backgroundColor = UIColor.gray
16. }

运行项目结果如图9所示。点击第二行单元格，在日志区会输出其对应行数，如图10、11所示。

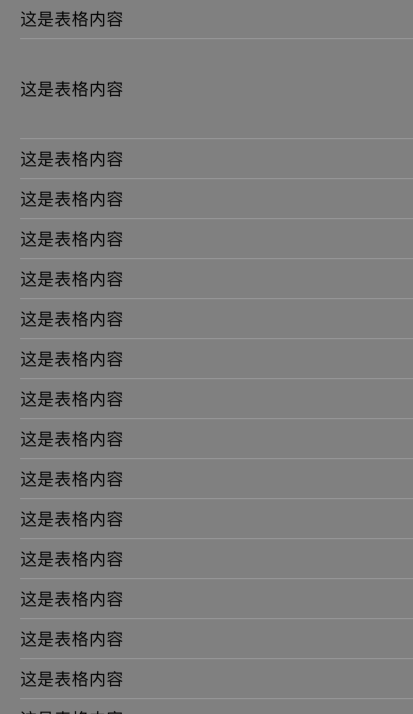
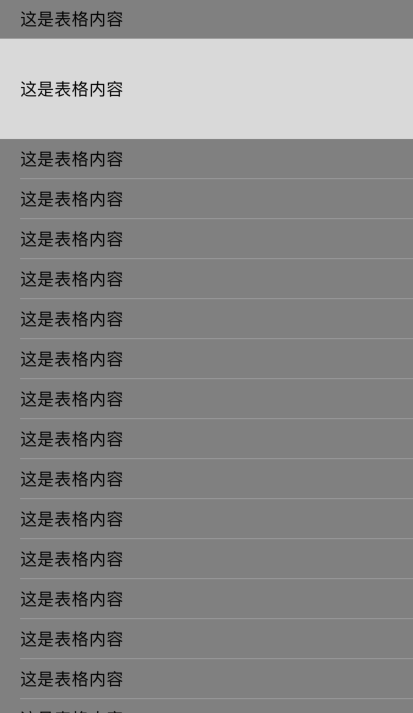
 

图9 单元格背景颜色改变 图10 点击第二行

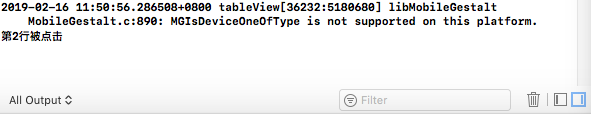


图11 日志输出结果

## 5.单元格的选择和取消选择

在手机应用程序中，单元格有时需要标记其被选中的状态，以便用户知晓曾经选中的目标。如对手机中的短信进行批处理的时候，在单元格右侧通常会有特殊的标记表示该单元格被选中，这样用户才能准确的对信息进行批处理。在UITableViewDelegate协议中就提供了简单的标记单元格被选中状态的方法，如tableView(\_:didSelectRowAt:)，tableView(\_:didDeselectRowAt:)，在第4节的实例中使用tableView(\_:didSelect RowAt:)简单实现了对点击单元格时实现输出单元格位置。本节就利用该方法标记单元格被选中和取消其选中的状态。

创建一个新项目，将ViewController.swift文件中的代码改写成如下代码。

1. import UIKit
2. class ViewController: UIViewController,UITableViewDataSource,
3. UITableViewDelegate{
4. var indexString=["第一行","第二行","第三行","第四行","第五行"]
5. override func viewDidLoad() {
6. super.viewDidLoad()
7. // Do any additional setup after loading the view, typically from a nib.
8. let tableRect = CGRect(x: 0, y: 20, width: UIScreen. main.bounds.width, height: UIScreen.main.bounds.height-20)
9. let tableView = UITableView(frame: tableRect)
10. tableView.dataSource=self
11. tableView.delegate=self
12. self.view.addSubview(tableView)
13. }
14. //设置UITableView表格行数
15. func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
16. return indexString.count
17. }
18. //单元格被点击时输出相应行数
19. func tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
20. let cell = tableView.cellForRow(at: indexPath)
21. if(cell?.accessoryType == UITableViewCell.AccessoryType.none){
22. cell?.accessoryType=UITableViewCell.AccessoryType.
23. checkmark
24. print("选择了第"+String(indexPath.row+1)+"行")
25. }
26. else{
27. cell?.accessoryType = UITableViewCell.AccessoryType.none
28. print("取消选择第"+String(indexPath.row+1)+"行")
29. }
30. }
31. //设置UITableViewCell单元格内容并使其能够复用
32. func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
33. let id="reused"
34. var cell = tableView.dequeueReusableCell(withIdentifier: id)
35. if(cell==nil){
36. cell=UITableViewCell(style: .default, reuseIdentifier: id)
37. }
38. cell?.textLabel?.text=indexString[indexPath.row]
39. return cell!
40. }
41. }

iOS系统的UITableViewCell.AccessoryType给出了5种单元格的附件类型，这些类型具体显示形式如表3所示。

表3 UITableViewCell.AccessoryType的类型

|  |  |
| --- | --- |
| 类型 | 图标 |
| UITableViewCell.AccessoryType.none | 无 |
| UITableViewCell.AccessoryType.checkmark | 屏幕快照 2019-02-16 下午3.40.33 |
| UITableViewCell.AccessoryType.DisclosureIndicator | 屏幕快照 2019-02-16 下午3.40.04 |
| UITableViewCell.AccessoryType.detailDisclosureButton | 屏幕快照 2019-02-16 下午3.40.21 |
| UITableViewCell.AccessoryType.detailButton | 屏幕快照 2019-02-16 下午3.40.49 |

在tableView(\_:didSelectRowAt:)方法中有两个参数：tableView和indexPath。其中tableview表示当前交互的表格对象，indexPath表示被选择的单元格所在索引位置。

在上述实例代码tableView(\_:didSelectRowAt:)方法中，通过表格对象的cellForRow(at:)方法得到被选择的单元格的位置，接着再判断该单元格的附件类型，根据其附件类型从而选择如何改变附件类型。运行项目，结果如图12、13所示。点击第二行，该单元格的附件类型改为.checkmark，并在日志区输出相应信息;再次点击，其附件类型变为.none，在日志区输出相应信息。

图12

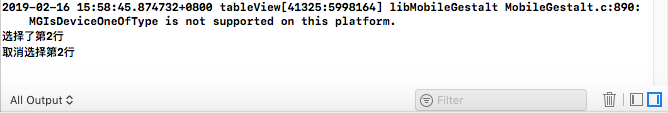


图13 日志输出结果

## UITableView的编辑

本节将要讲解对单元格的添加、删除和位置的改变。单元格的添加和删除要求通过表格对象来对数据源中的数据进行操作，单元格位置的改变是通过拖动单元格将其移动到另一个位置。

接下来将实现单元格的简单添加和删除操作。

1. import UIKit
2. class ViewController: UIViewController,UITableViewDataSource, UITableViewDelegate{
3. var dic:Dictionary<Int,[String]>!
4. override func viewDidLoad() {
5. super.viewDidLoad()
6. // Do any additional setup after loading the view, typically from a nib.
7. dic = [
8. 0:[String](["11","22","33","44","55"]),
9. 1:[String](["111","222","333","444"])
10. ]
11. let tableRect = CGRect(x: 0, y: 20, width: UIScreen.main.bounds.width, height: UIScreen.main.bounds.height-20)
12. let tableView = UITableView(frame: tableRect)
13. tableView.dataSource=self
14. tableView.delegate=self
15. //使UITableView进入编辑模式
16. tableView.setEditing(true, animated: true)
17. self.view.addSubview(tableView)
18. }
19. //设置UITableView表格行数
20. func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
21. return dic[section]!.count+1
22. }
23. //设置UITableView表格章节数
24. func numberOfSections(in tableView: UITableView) -> Int {
25. return dic.count
26. }
27. //设置每一章节标题
28. func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
29. return String(section)
30. }
31. //设置各单元格的编辑图标类型
32. func tableView(\_ tableView: UITableView, editingStyleForRowAt indexPath: IndexPath) -> UITableViewCell.EditingStyle {
33. if(indexPath as NSIndexPath).row == dic[indexPath.section]?.count{
34. return UITableViewCell.EditingStyle.insert
35. }
36. return UITableViewCell.EditingStyle.delete
37. }
38. //设置点击编辑图标后的动作
39. func tableView(\_ tableView: UITableView, commit editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
40. //对键值对和表格都进行插入操作
41. if editingStyle == UITableViewCell.EditingStyle.delete{
42. dic[indexPath.section]?.remove(at: indexPath.section)
43. tableView.deleteRows(at: [indexPath], with: .fade)
44. }
45. //对键值对和表格都进行删除操作
46. if editingStyle == UITableViewCell.EditingStyle.insert{
47. dic[indexPath.section]?.insert(String(indexPath.row+1), at: indexPath.row)
48. tableView.insertRows(at: [indexPath], with: .fade)
49. }
50. }
51. //设置UITableViewCell单元格内容并使其能够复用
52. func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
53. let id="reused"
54. var cell = tableView.dequeueReusableCell(withIdentifier: id)
55. if(cell==nil){
56. cell=UITableViewCell(style: .default, reuseIdentifier: id)
57. }
58. if (indexPath as NSIndexPath).row == dic[indexPath.section]?.count{
59. cell?.textLabel?.text = "添加一行数据"
60. }
61. else{
62. cell?.textLabel?.text=dic[indexPath.section]![indexPath.row]
63. }
64. return cell!
65. }
66. }

上述代码通过<int,[string]>类型的键值对作为数据源对表格内容进行编辑。其中需要注意的是第16行的setEditing(animated:)方法，调用该方法，并将其第一个参数值设为true，UITableView才能进入编辑模式，单元格才能被编辑。该方法有两个枚举参数：editing,当值为true时，对应的UITableView进入编辑模式，单元格可以显示其编辑图标和排序图标，默认值为false，即非编辑模式，图标不显示；animated，当值为true时，对应的UITableView进入编辑模式时有动画，默认值为false，即没有动画直接进入编辑模式。图14展示了非编辑模式和编辑模式下单元格的区别。编辑模式下单元格的左侧会空出空间给编辑图标，右侧空间给排序图标。

非编辑模式 编辑模式

非编辑模式 编辑模式

图14

tableView(\_:editingStyleForRowAt:)方法设置各个单元格显示删除图标还是插入图标，单元格的图标由该方法的返回值来决定。对于每个单元格，在tableView进入编辑模式后都会运行该方法。编辑类型editingStyle有3个枚举类型：.none，单元格不显示任何编辑图标，点击单元格无任何效果；.insert，单元格显示插入图标，点击该图标插入一行单元格；.delete，为默认样式，单元格显示删除图标，点击该图标单元格左滑,在右侧显示其默认的带有Delete字样的附件按钮，点击该按钮删除一行单元格。

tableView(\_:commit:,forRowAt:)方法通过识别单元格的editingStyle，设置点击编辑图标对应的动作。在上述代码中，点击的单元格编辑图标的editingStyle为.insert时，先在tableView对应的数据源插入一条数据，保持tableView与数据源的数据的一致性，再调用tableView的insertRows(\_:at:)方法，在指定索引位置插入一条新的单元格。同理editingStyle为.delete时，先删除数据源对应的数据，再删除对应索引位置的单元格。

insertRows(\_:at:)和deleteRows(\_:at:)都有两个相同的枚举类型参数：at和animation。其中at参数是indexPath对象构成的数组，也就是说可以通过一个含有多个indexPath对象的数组实现对单元格的批量处理。animation参数用来设置当单元格被插入或删除时所执行的动画格式。动画格式的类型及其动画效果如表4所示。

表4 animation的枚举成员

|  |  |
| --- | --- |
| 成员 | 说明 |
| .fade | 插入或删除单元格以淡入方式显示或以淡出方式移除 |
| .left | 待插入或删除单元格从左侧滑入或滑出 |
| .right | 待插入或删除单元格从右侧滑入或滑出 |
| .top | 待插入或删除单元格从底部滑入或滑出 |
| .bottom | 待插入或删除单元格从顶部滑入或滑出 |
| .none | 待插入或删除单元格以默认方式显示或移除 |
| .middle | 尽量使新旧单元格居中显示在其曾经或将要显示的位置 |
| .automatic | 系统自行选择合适的动画方式 |

运行项目，结果如图15所示。点击单元格的删除图标和插入图标，观察运行效果。点击0章节的插入图标，效果如图16所示；点击删除图标时，效果如图17所示。

图15 图16

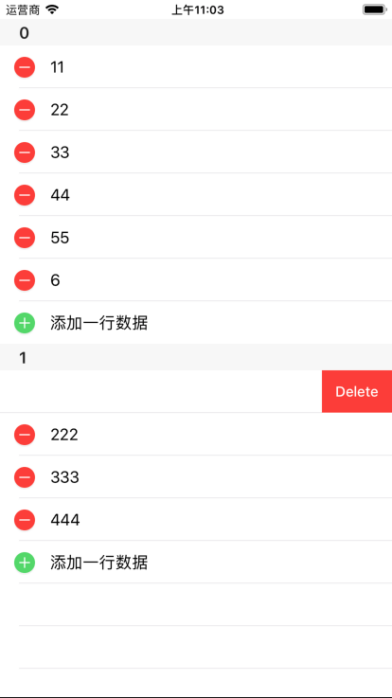
 

图17

对于单元格位置的改变，具体要实现的操作是拖拽单元格右侧的拖拽图标到指定位置，以此实现位置的改变。以下示例实现创建表格并使单元格可以拖拽改变位置。

1. import UIKit
2. class ViewController: UIViewController,UITableViewDataSource,UITableViewDelegate{
3. var dic:Dictionary<Int,[String]>!
4. override func viewDidLoad() {
5. super.viewDidLoad()
6. // Do any additional setup after loading the view, typically from a nib.
7. dic = [
8. 0:[String](["11","22","33","44","55"]),
9. 1:[String](["111","222","333","444"])
10. ]
11. let tableRect = CGRect(x: 0, y: 20, width: UIScreen.main.bounds. width, height: UIScreen.main.bounds.height-20)
12. let tableView = UITableView(frame: tableRect)
13. tableView.dataSource=self
14. tableView.delegate=self
15. //使UITableView进入编辑模式
16. tableView.setEditing(true, animated: false)
17. self.view.addSubview(tableView)
18. }
19. func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
20. return dic[section]!.count
21. }
22. func numberOfSections(in tableView: UITableView) -> Int {
23. return dic.count
24. }
25. func tableView(\_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
26. return String(section)
27. }
28. //此处设置返回值为.none
29. func tableView(\_ tableView: UITableView, editingStyleForRowAt indexPath: IndexPath) -> UITableViewCell.EditingStyle {
30. return UITableViewCell.EditingStyle.none
31. }
32. //设置拖拽单元格的具体动作
33. func tableView(\_ tableView: UITableView, moveRowAt sourceIndexPath: IndexPath, to destinationIndexPath: IndexPath) {
34. let source = sourceIndexPath.row
35. let destination = destinationIndexPath.row
36. let content = dic[sourceIndexPath.section]![sourceIndexPath.row]
37. dic[sourceIndexPath.section]?.remove(at: source)
38. if destinationIndexPath.row > (dic[destinationIndexPath.section]?.count)!{
39. dic[destinationIndexPath.section]?.append(content)
40. }
41. else{
42. dic[destinationIndexPath.section]?.insert(content, at: destination)
43. }
44. }
45. //设置UITableViewCell单元格内容并使其能够复用
46. func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
47. let id="reused"
48. var cell = tableView.dequeueReusableCell(withIdentifier: id)
49. if(cell==nil){
50. cell=UITableViewCell(style: .default, reuseIdentifier: id)
51. }
52. cell?.textLabel?.text=dic[indexPath.section]![indexPath.row]
53. return cell!
54. }
55. }

运行项目，长按单元格右侧的图标，拖拽单元格到其他位置实现单元格位置的改变，如图18所示。

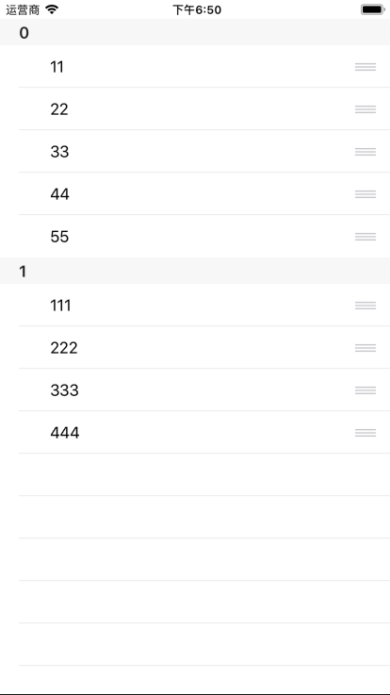
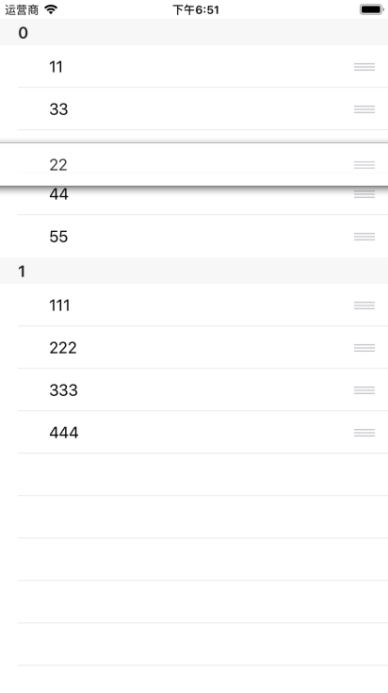
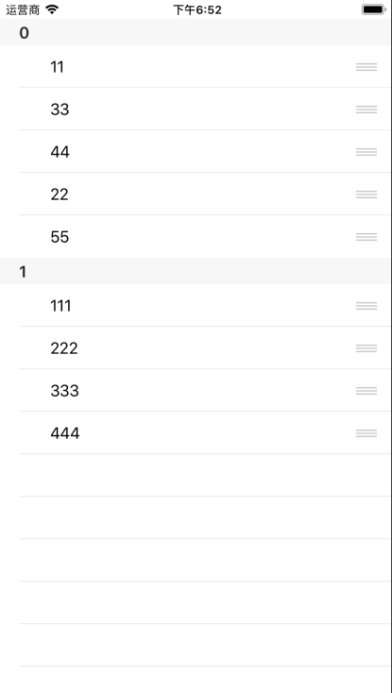
  

图18

由于只需实现对单元格位置的改变，tableView(\_:editingStyleForRowAt:)方法的返回值设为.none。而tableView(\_:moveRowAt:to:)方法实现了拖拽动作的具体执行，当前被拖动的单元格索引位置由sourceIndexPath参数表示，拖拽结束单元格要停留的索引位置由destinationIndexPath参数表示。

以上代码实现的结果可以使得单元格从一个分区移动到另一个分区，如把第一个section的单元格移到第二个section中。如果想要实现单元格只能在自己原分区中移动，在上述代码的第44行后添加以下代码。

1. //设置单元格只能在自己分区移动
2. func tableView(\_ tableView: UITableView, targetIndexPathForMoveFromRowAt sourceIndexPath: IndexPath, toProposedIndexPath proposedDestinationIndexPath: IndexPath) -> IndexPath {
3. if sourceIndexPath.section != proposedDestinationIndexPath.section{
4. var row = 0
5. //如果向上面的分区拖拽，单元格最终位置为原分区开头
6. //如果向下面的分区拖拽，单元格最终位置为原分区结尾
7. if sourceIndexPath.section < proposedDestinationIndexPath.section{
8. row = tableView.numberOfRows(inSection: sourceIndexPath.section)-1
9. }
10. return IndexPath(row: row, section: sourceIndexPath.section)
11. }
12. return proposedDestinationIndexPath
13. }

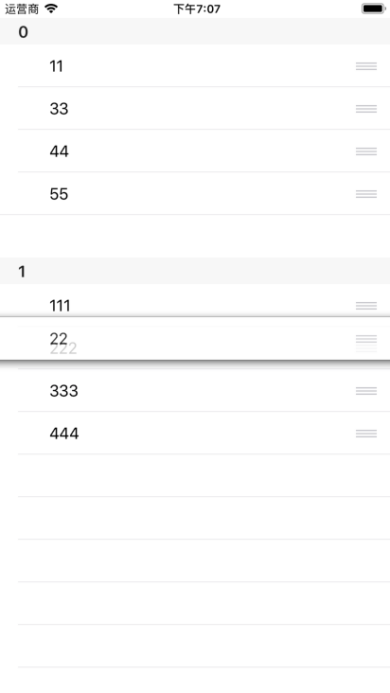
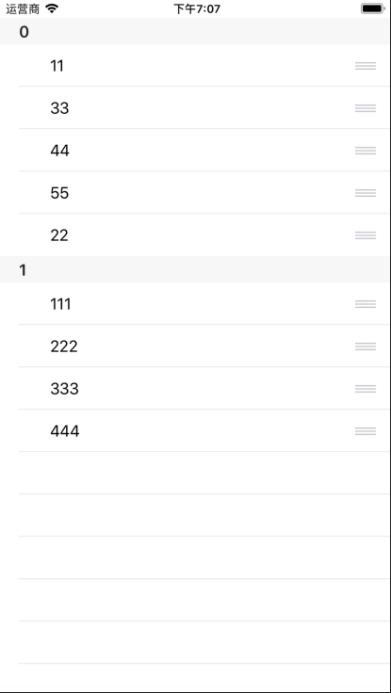
再次运行项目，拖动第一个section的单元格至第二个单元格，结果如图19所示。 

图19

实际上当用户拖动单元格右侧的拖拽按钮时，被拖动的单元格在滑过每个单元格时都需要调用tableView(\_:targetIndexPathForMoveFromRowAt:toProposedIndexPath)方法来判断当前滑过的单元格是否可以与被拖拽的单元格进行替换，默认返回参数propose- dDestinationIndexPath。当用户结束对单元格的拖动时，调用tableView(\_:moveRowAt:to:)方法实现最终被拖拽的单元格位置变换。

## 小结

本章节讲解了对UITableView的简单使用，包括表格的创建、自定义单元格、单元格被选择时的标记、对表格内容的编辑。UITableView在应用程序中的用途广泛，是iOS开发中必不可少的，因此本章内容需重点掌握。