

第六部分 表格



授课教师：杨文川



表格（2学时）

表格视图是非常重要的一个视图控件，常常用来展示复杂的表格信息，几乎所有的苹果应用都会用到这个控件。

本章目的在于使学生掌握表格视图的基本概念和高级属性，包括：编辑表格视图、定制单元格、搜索栏、以及视图的分节和刷新等。每一个重要的功能点都会通过迭代的方式添加到最初的表格视图实例中，并将新的知识点通过新增功能加入到实例中。

- 表格视图
- 编辑表格视图
- 表格视图单元格
- 表格视图刷新

表格视图

- 表格视图UITableView是一种用来按行显示数据的视图。UITableView是UIScrollView的子类，支持垂直方面的页面滚动（不支持横向滚动）。



- UITableView的每行为一个基本的组成单元格，即：*UITableViewCell*
- UITableView的显示是通过UITableViewCell将每一行的单元格内容画出来的。
- 每一行的单元格Cell都含有标题、图片，还可以有附属视图。
- 标准的附属视图是扩展指示器或者详情扩展按钮。
- 扩展指示器将会展开下一级数据，而详情扩展按钮则会打开该行的详细信息页视图。表格视图可以进入编辑模式，在此模式下，用户可以对单元格进行插入、删除以及保存的操作。

表格视图的分节

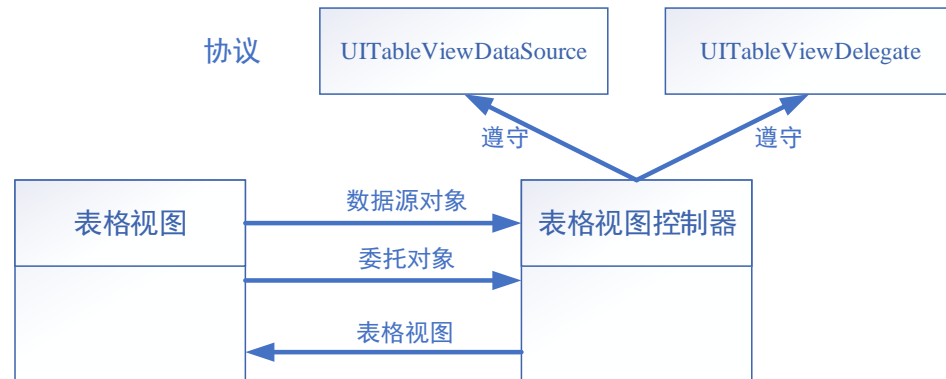
- 表格视图可以分节显示，也可以不分节。
- 每个分节由若干个单元格组成。
- 表格视图中的sectors属性决定了一个表格视图有多少的分节，分节的单元格行数rows属性决定了每个分节有多少行单元格。表格视图的任意分节都可以由一个header和footer。
- 表格视图有两种显示风格：普通（plain）和分组（grouped）。
 - 普通风格下，节头和节脚在分节内容上浮动显示。如果表格视图有索引，那么在表格视图的右侧将会出现一个检索栏，通过该检索栏可以快速定位到相应的分节处。
 - 分组风格下，所有的单元格都有一个缺省的背景色和背景视图。通过背景视图可以在同一分节中将单元格进行分组。



表格视图的对象关系图

- 每一个表格视图UITableView对象都必须有一个对应的数据源对象和委托对象，通常由该表格视图的控制器UITableViewController对象来担任。
- 数据源对象必须遵守协议UITableViewDataSource，委托对象必须遵守协议UITableViewDelegate。
- 数据源对象向表格视图提供表格在创建和编辑单元格时需要的数据模型信息。
- 委托对象负责表格视图中单元格的配置、选择、保存、附属视图以及编辑操作。

◆ 在表格视图控制器UITableViewController创建其表格视图UITableView时，视图的数据源和委托会自动的指向UITableViewController，不需要手动设置。





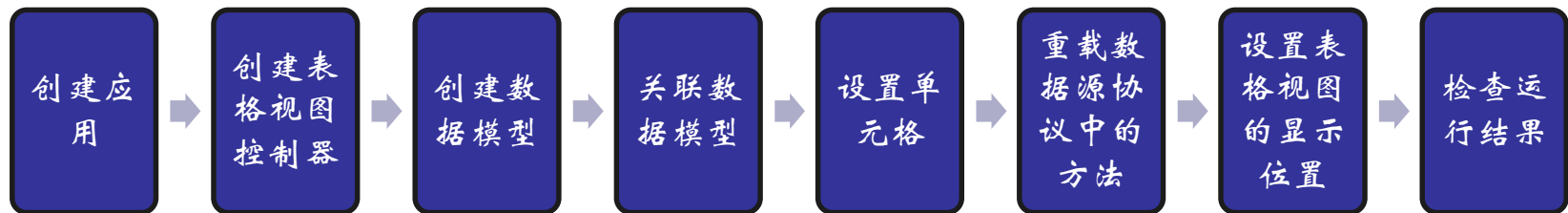
实例：学生成绩表1.0

- 本例将通过UITableView来创建一个学生成绩表。
- **要求**
 - 应用的界面上为一个表格视图，用来表示成绩单，成绩单由学生名字和成绩两部分信息的列表组成。

代码参见iosPrj的Chapter05-
StudentPerformance

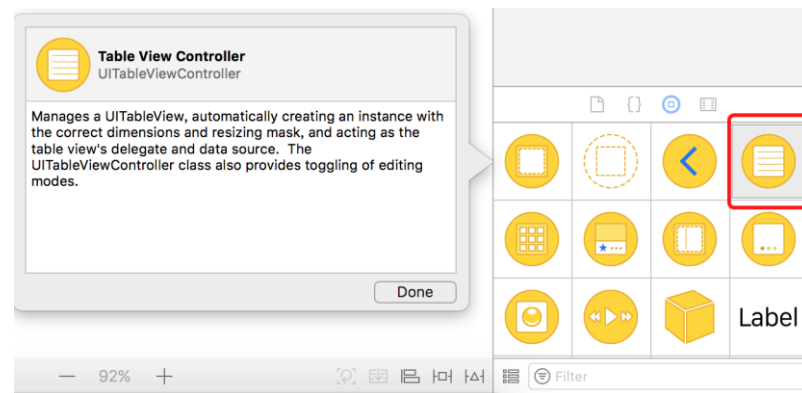


应用开发流程



创建表格视图控制器

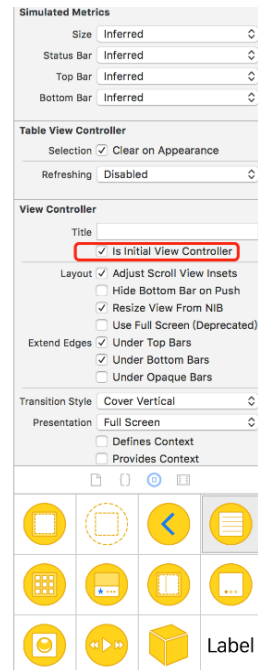
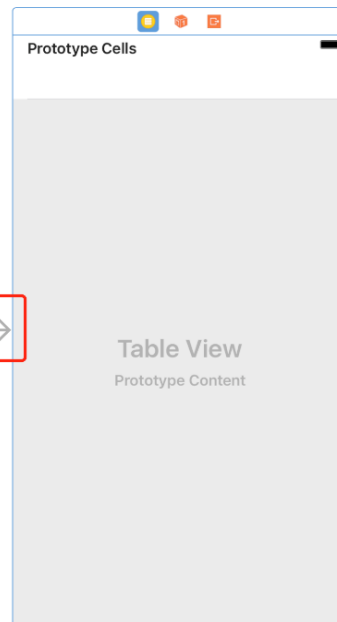
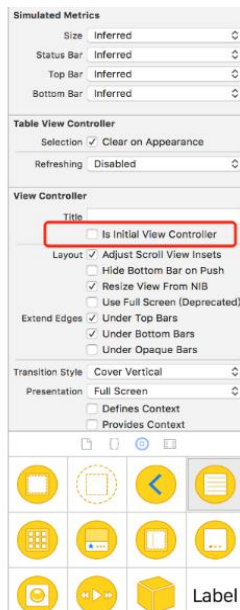
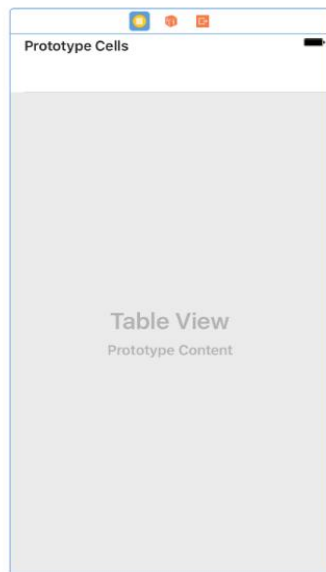
- 通过single view application模板来创建工程StudentPerformance。
- 打开故事板main.storyboard，删除其中缺省的View Controller。
- 然后在控件库中找到Table View Controller，并将其拖拽到中间的视图编辑面板中



设置初始视图控制器

- 项目缺省的初始视图控制器是刚刚删除的View Controller，因此需要指定新创建的Table View Controller为初始视图控制器

将Table View Controller设置为初始视图控制器后，视图左侧出现了一个箭头，表示该视图为项目启动后进入的初始视图。



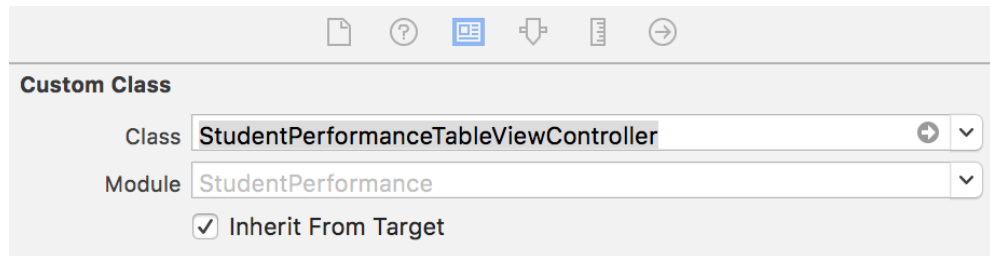
创建表格视图控制器文件

- 为Table View Controller实例创建相应的视图控制器类文件。
 - 先删除项目缺省的视图控制器文件ViewController.swift，然后创建一个新的swift文件。
 - 该类在本项目中将承担三个角色
 - 表格视图控制器，负责表格视图的显示；
 - 表格视图数据源，负责向表格视图提供显示用到的数据；
 - 表格视图的委托，负责表格视图相关事件处理和通讯。
- 创建完表格视图控制器类后，要将其指定为故事板中表格视图控制器实例的类

```
import UIKit

class StudentPerformanceTableViewController: UITableViewController {

}
```



代码参见iosPrj的Chapter05-
StudentPerformance

创建数据模型

- 表格中显示的是学生姓名及其考试成绩，因此创建学生类 `student`。
- 表格中的每一行对应一个学生，也就是 `Student` 类的一个实例。
- 用 `Student` 类来抽象表格视图中的单元格数据，由 `Student` 类实例组成的集合就可以用来抽象整个表格的数据。
- 创建学生信息类 `StudentsInfo`。该类只有一个变量 `studentsCollection`，是 `Student` 类的数组，用来保存所有学生实例的信息。

代码参见 `iosPrj` 的 `Chapter05-StudentPerformance`

```
import UIKit

class Student: NSObject {

    var name: String
    var score: Int

    init(name: String, score: Int) {

        self.name = name
        self.score = score
        super.init()

    }

}

import UIKit

class StudentsInfo {

    var studentsCollection = [Student]()

    init() {

        let nameOfStudents =
            ["Tommy", "Jerry", "Kate", "Jack", "Ben", "Jimmy", "Ada", "Susan"]

        let scoreOfStudents = [98, 65, 78, 100, 85, 79, 95, 80, 83]

        for i in 0...(nameOfStudents.count-1) {

            let theStudent = Student(name: nameOfStudents[i], score:
                scoreOfStudents[i])

            studentsCollection.append(theStudent)

        }

    }

}
```



关联数据模型

- 在表格视图控制器定义中声明一个StudentsInfo类的变量，用来接收相关的数据模型数据。
- 在AppDelegate.swift文件的相应方法中添加代码。该方法在应用即将呈现给用户时执行，可以在此方法中进行最后的初始化工作。

代码参见iosPrj的Chapter05-StudentPerformance

```
import UIKit

class StudentPerformanceTableViewController:
    UITableViewController {

    var studentsInfo: StudentsInfo!

func application(_ application: UIApplication, didFinishLaunchingWithOptions
    launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.

    let studentsInfo = StudentsInfo()

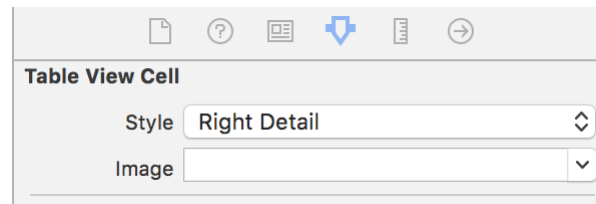
    let studentPerformanceTableViewController = window!.rootViewController
        as! StudentPerformanceTableViewController

    studentPerformanceTableViewController.studentsInfo = studentsInfo

    return true
}
```

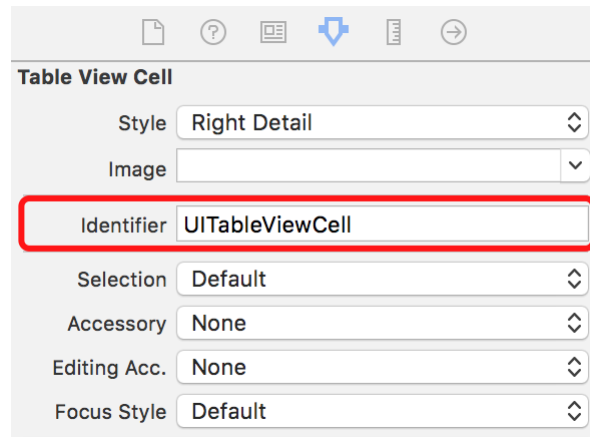
设置单元格的显示风格

- 根据应用的要求，每一个单元格要显示两个信息：姓名和对应的成绩。
- 选中视图面板中的表格视图控制器，然后在右侧的属性设置面板中将Table View Cell的Style属性设置为Right Detail。
- 设置完以后，就可以看到表格视图中的单元格显示风格变为左侧显示title，右侧显示detail。



设置单元格的属性

- 打开视图面板，选中Table View Cell，在其右侧的属性面板中设置Identifier为UITableViewCell。





重载表格视图数据源方法

- 重载协议
UITableViewDataSource中的方法

- tableView(_:numberOfRowsInSection:)

- 该方法通知UITableView一共有多少行单元格需要显示。

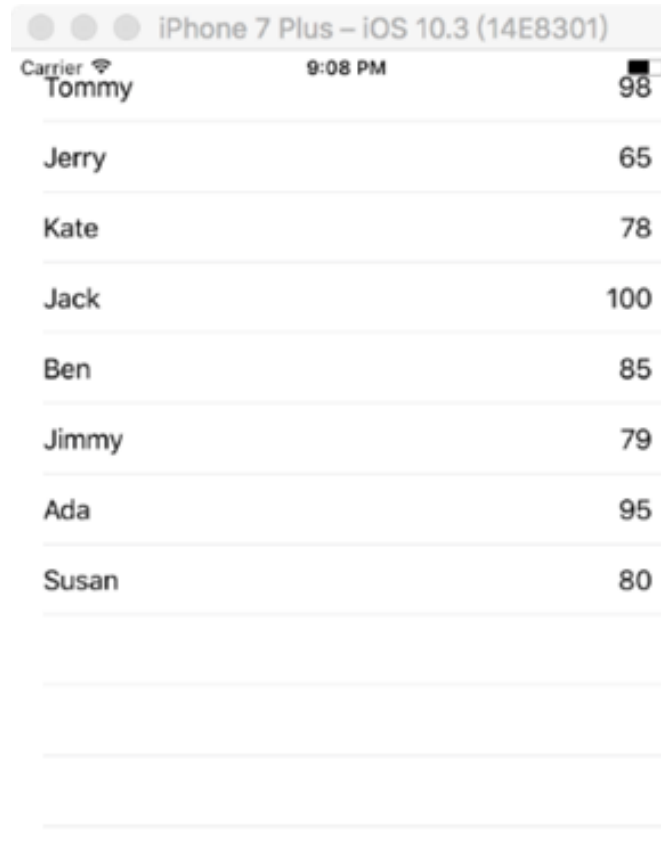
- tableView(_:cellForRowAt:)

- 该方法通知UITableView，需要显示的每一个单元格的内容。
- UITableView的方法 dequeueReusableCell(withIdentifier:) 将会在内存中找 Identifier 属性为 UITableViewCell 的单元格，如果找到就重用该单元格，如果没有找到，则按照该单元格创建一个。这种机制可以重用单元格资源，从而节省大量的内存资源。

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection: Int) -> Int {  
  
    return studentsInfo.studentsCollection.count  
}
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
  
    let cell = tableView.dequeueReusableCell(withIdentifier: "UITableViewCell", for: indexPath)  
  
    let student = studentsInfo.studentsCollection[indexPath.row]  
  
    cell.textLabel?.text = student.name  
    cell.detailTextLabel?.text = "\(student.score)"  
  
    return cell  
}
```

运行效果：表格视图与状态栏重叠



The screenshot shows an iPhone 7 Plus screen with a table of names and scores. The table is overlaid on the status bar, which is visible at the top of the screen. The status bar shows the carrier 'Tommy', the time '9:08 PM', and the battery level '98'. The table has a header row with 'Jerry', 'Kate', 'Jack', 'Ben', 'Jimmy', 'Ada', and 'Susan'. The scores are 65, 78, 100, 85, 79, 95, and 80 respectively. The table is overlaid on the status bar, which is visible at the top of the screen.

Jerry	65
Kate	78
Jack	100
Ben	85
Jimmy	79
Ada	95
Susan	80



设置表格视图的显示位置

- 重载viewDidLoad()方法，在该方法中添加代码。
 - 取得系统状态栏的高度
 - 将该高度设置为UITableView的顶点的top值，其它值不变。
- 将UITableView整体平移到状态栏下面，这样就不会发生重叠了。

```
override func viewDidLoad() {  
  
    let statusBarHeight = UIApplication.shared.statusBarFrame.height  
  
    let insets = UIEdgeInsets(top: statusBarHeight, left: 0, bottom: 0,  
                              right: 0)  
  
    tableView.contentInset = insets  
  
    tableView.scrollIndicatorInsets = insets  
  
}
```

[illegible]



第六部分 表格

表格视图是非常重要的一个视图控件，常常用来展示复杂的表格信息，几乎所有的苹果应用都会用到这个控件。

本章目的在于使学生掌握表格视图的基本概念和高级属性，包括：编辑表格视图、定制单元格、搜索栏、以及视图的分节和刷新等。每一个重要的功能点都会通过迭代的方式添加到最初的表格视图实例中，并将新的知识点通过新增功能加入到实例中。

- 表格视图
- 编辑表格视图
- 表格视图单元格
- 表格视图刷新



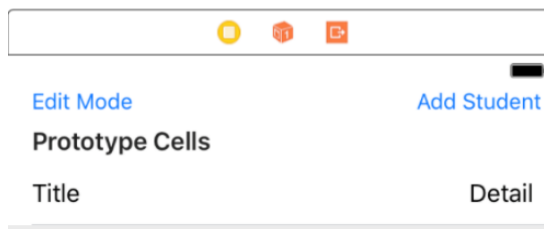
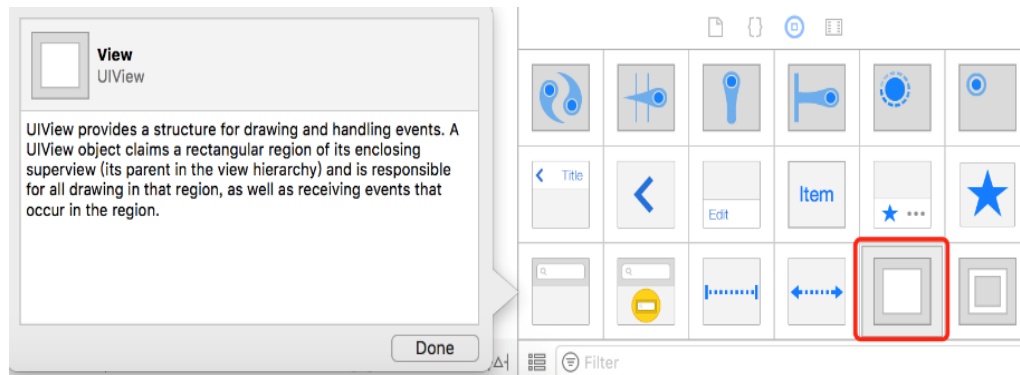
编辑表格视图

- 表格视图UITableView的属性editing用来表示表格视图是否处于编辑状态
- 当处于编辑状态时editing为true，否则为false。
- 当表格视图进入到编辑状态时，就可以对表格视图中的单元格进行操作。
- 本节主要介绍最常见的几个操作的实现方法
 - 添加单元格
 - 删除单元格
 - 移动单元格。

代码参见iosPrj的Chapter05-
StudentPerformance2.0

定制单元格

- 选中故事板，从空间库中拖拽一个UIView到表格视图中的单元格Prototype Cells上面，也就是表头的位置
- 从组件库中拖拽两个UIButton，分别放置在UIView的左右两端，并将其title分别修改为“Edit mode”和“Add Student”
- 建立这两个按钮与表格视图控制器的连接关系



```
@IBAction func addStudent(_ sender: UIButton) {  
}  
  
@IBAction func shiftEditMode(_ sender: UIButton) {  
}
```

切换表格视图编辑模式的方法

- `isEditing`是 `UITableViewController` 的表示是否处于编辑状态的布尔变量。
- 如果`isEditing`为`true`，则当前已处于编辑状态，此时点击切换按钮则应该重新进入非编辑状态。
- 这里通过方法`setEditing`来设置编辑状态的属性。
- 当编辑状态切换的时候，修改按钮的标题增加了应用的用户友好性。

```
@IBAction func shiftEditMode(_ sender: UIButton) {  
    if isEditing {  
        sender.setTitle("Edit Mode", for: .normal)  
        setEditing(false, animated: true)  
    } else {  
        sender.setTitle("Submit", for: .normal)  
        setEditing(true, animated: true)  
    }  
}
```

Carrier 12:55 AM

Submit Add Student

—	Tommy	98	
ry		65	Delete
—	Kate	78	
—	Jack	100	
—	Ben	85	
—	Jimmy	79	
—	Ada	95	
—	Susan	80	

代码参见iosPrj的Chapter05-
StudentPerformance2.0



按钮Add Student的动作处理函数

- 点击按钮Add Student后，需要向表格中增加一条学生成绩记录，分两步来实现：
 - 在StudentsInfo类中定义一个新的函数addStudent，用来创建一个新的学生成绩记录，并将其添加到学生记录数组中，并返回这条学生成绩记录。
 - 编写按钮Add Student的动作处理函数。首先，创建一个学生成绩对象。然后，根据该对象在学生数组中的序号，添加到表格视图中相应的位置。

```
func addStudent() -> Student {  
  
    let theStudent = Student(name: "The Student", score: 100)  
  
    studentsCollection.append(theStudent)  
  
    return theStudent  
}  
  
@IBAction func addStudent(_ sender: UIButton) {  
  
    let theStudent = studentsInfo.addStudent()  
  
    if let theIndex = studentsInfo.studentsCollection.index(of:  
        theStudent) {  
  
        let theIndexPath = IndexPath(row: theIndex, section: 0)  
  
        tableView.insertRows(at: [theIndexPath], with: .  
            automatic)  
  
    }  
}
```

代码参见iosPrj的Chapter05-
StudentPerformance2.0



运行效果

iPhone 7 Plus – iOS 10.3 (14E8301)

Carrier 1:15 AM

[Edit Mode](#) [Add Student](#)

Tommy	98
Jerry	65
Kate	78
Jack	100
Ben	85
Jimmy	79
Ada	95
Susan	80
The Student	100
The Student	100
The Student	100
The Student	100
The Student	100



实现删除单元格的功能

- 在StudentsInfo类中增加一个删除学生成绩记录的函数
- 在表格视图控制器对象中重载数据源协议中的相应方法。该方法在插入或删除单元格的时候触发。

```
func deleteStudent(_ theStudent: Student) {  
    if let theIndex = studentsCollection.index(of: theStudent) {  
        studentsCollection.remove(at: theIndex)  
    }  
}  
  
override func tableView(_ tableView: UITableView, commit editingStyle:  
    UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {  
    if editingStyle == .delete {  
        let theStudent = studentsInfo.studentsCollection[indexPath.row]  
        studentsInfo.deleteStudent(theStudent)  
        tableView.deleteRows(at: [indexPath], with: .automatic)  
    }  
}
```

代码参见iosPrj的Chapter05-
StudentPerformance2.0



实现移动单元格的功能

- 在StudentsInfo类中增加移动数组记录所在位置的方法。

```
func transferPosition(sourceIndex: Int, destinationIndex: Int) {  
    if sourceIndex != destinationIndex {  
        let theStudent = studentsCollection[sourceIndex]  
        studentsCollection.remove(at: sourceIndex)  
        studentsCollection.insert(theStudent, at: destinationIndex)  
    }  
    return  
}
```

- 在表格视图控制器对象中重载协议
UITableViewDataSource
中的方法
tableView(UITableView,
moveRowAt: IndexPath,
to: IndexPath)。

```
override func tableView(_ tableView: UITableView, moveRowAt  
sourceIndexPath: IndexPath, to destinationIndexPath: IndexPath) {  
    studentsInfo.transferPosition(sourceIndex: sourceIndexPath.row,  
destinationIndex: destinationIndexPath.row)  
}
```



第六部分 表格

表格视图是非常重要的一个视图控件，常常用来展示复杂的表格信息，几乎所有的苹果应用都会用到这个控件。

本章目的在于使学生掌握表格视图的基本概念和高级属性，包括：编辑表格视图、定制单元格、搜索栏、以及视图的分节和刷新等。每一个重要的功能点都会通过迭代的方式添加到最初的表格视图实例中，并将新的知识点通过新增功能加入到实例中。

- 表格视图
- 编辑表格视图
- 表格视图单元格
- 表格视图刷新

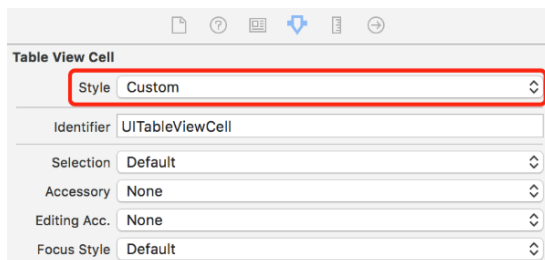
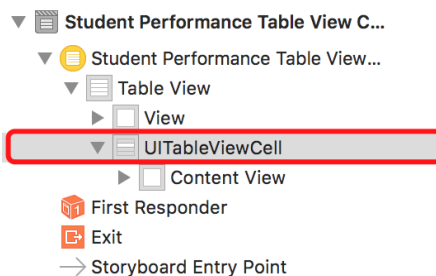


表格视图单元格

- 表格视图单元格类UITableViewController负责设置单元格的内容、背景（包括文本、图片以及自定义视图）、状态、附属视图以及单元格内容编辑的初始化。
- 当表格视图创建单元格时，既可以使用系统缺省的风格也可以自定义单元格。系统预先定义好的缺省的单元格风格是不能改变的，包括其中的子视图的位置和尺寸等。使用这种缺省的单元格风格，只需要提供内容即可。
- 对于大多数应用来说，表格视图的单元格缺省提供的textLabel、detailTextLabel、imageView就能满足需要了。当不能满足时，就需要通过继承UITableViewController的方式来自定义单元格的内容。
- 自定义单元格视图时，要手动添加需要的子视图控件。
 - 注意：不能将子视图直接添加到单元格上，而是要添加到单元格的contentView上。
 - contentView在程序运行的时候，会根据情况自动调节尺寸，不会出现显示异常。

自定义单元格实例：扩展一个新的字段信息

- 定义一个单元格的类
- 关联StudentCell类和故事板中的Cell视图



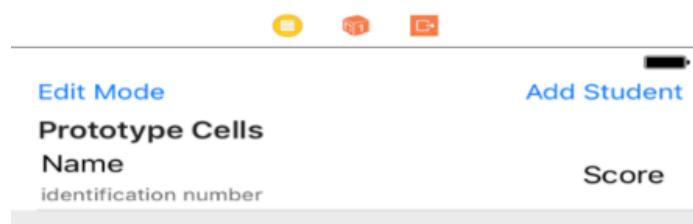
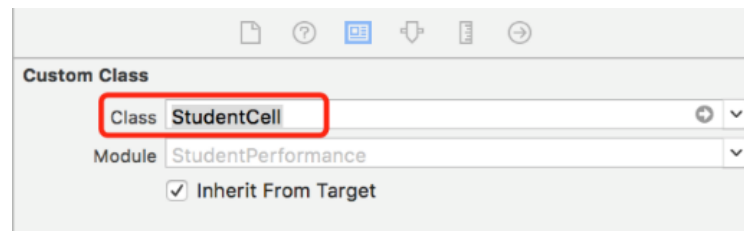
- 将单元格的三个UILabel连接到StudentCell类

代码参见iosPrj的Chapter05-StudentPerformance3.0

```
import UIKit

class StudentCell: UITableViewCell {

}
```



```
import UIKit

class StudentCell: UITableViewCell {

    @IBOutlet var nameLabel: UILabel!

    @IBOutlet var scoreLabel: UILabel!

    @IBOutlet var idLabel: UILabel!

}
```

修改数据模型

- 增加一个字段 “学号”

◆ 修改StudentInfo类

```
import UIKit

class Student: NSObject {

    var name: String
    var score: Int
    var id: String

    init(name: String, score: Int, id: String) {

        self.name = name
        self.score = score
        self.id = id
        super.init()

    }

}
```

代码参见iosPrj的Chapter05-
StudentPerformance3.0

```
init() {

    let nameOfStudents =
        ["Tommy", "Jerry", "Kate", "Jack", "Ben", "Jimmy", "Ada", "Susan"]

    let scoreOfStudents = [98, 65, 78, 100, 85, 79, 95, 80, 83]

    let idOfStudents =
        ["37060101", "37060102", "37060103", "37060104", "37060105", "37060106",
        "37060107", "37060108"]

    for i in 0...(nameOfStudents.count-1) {

        let theStudent = Student(name: nameOfStudents[i], score:
            scoreOfStudents[i], id: idOfStudents[i])

        studentsCollection.append(theStudent)

    }

}

func addStudent() -> Student {

    let theStudent = Student(name: "The Student", score: 100, id:
        "37060109")

    studentsCollection.append(theStudent)

    return theStudent

}
```



修改数据模型

- 修改表格视图控制器中用来显示单元格内容的 UITableViewDataSource 协议方法 tableView(_ tableView: UITableView, cellForRowAt indexPath: NSIndexPath)

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
  
    let cell = tableView.dequeueReusableCell(withIdentifier: "UITableViewCell", for: indexPath) as! StudentCell  
  
    let student = studentsInfo.studentsCollection[indexPath.row]  
  
    cell.nameLabel.text = student.name  
    cell.idLabel.text = student.id  
    cell.scoreLabel.text = "\(student.score)"  
  
    return cell  
}
```

[illegible]



第六部分 表格

表格视图是非常重要的一个视图控件，常常用来展示复杂的表格信息，几乎所有的苹果应用都会用到这个控件。

本章目的在于使学生掌握表格视图的基本概念和高级属性，包括：编辑表格视图、定制单元格、搜索栏、以及视图的分节和刷新等。每一个重要的功能点都会通过迭代的方式添加到最初的表格视图实例中，并将新的知识点通过新增功能加入到实例中。

- 表格视图
- 编辑表格视图
- 表格视图单元格
- 表格视图刷新



表格视图刷新

- 表格视图的下拉刷新是苹果应用中常见的功能。
- 在iOS系统中刷新功能可以通过控件UIRefreshControl来实现。
- UITableViewController本身就自带refreshControl属性，缺省值为nil。
- 当要使用刷新控件时，直接设置表格视图的刷新控件的属性即可。设置完毕，刷新控件会在表格视图的顶部出现。
- 下面，我们为前面的例子增加数据刷新的功能
 - 只需进行两处修改



定制刷新控件

- 在方法viewDidLoad()中添加创建UIRefreshControl的代码
 - 创建一个刷新控件的实例；
 - 设置刷新控件的属性，即：下拉的时候显示的标题；
 - 刷新控件绑定事件的处理函数；
 - 将新定制的刷新控件赋值给视图控制器的刷新控件。

```
override func viewDidLoad() {  
    let statusBarHeight = UIApplication.shared.statusBarFrame.  
        height  
  
    let insets = UIEdgeInsets(top: statusBarHeight, left: 0,  
        bottom: 0, right: 0)  
  
    tableView.contentInset = insets  
  
    tableView.scrollIndicatorInsets = insets  
  
    //add refreshControl into view  
    let theRefreshControl = UIRefreshControl()  
  
    theRefreshControl.attributedTitle =  
        NSAttributedString(string: "refreshing")  
  
    theRefreshControl.addTarget(self, action: #selector  
        (refreshing), for: UIControlEvents.valueChanged)  
  
    refreshControl = theRefreshControl  
}
```

代码参见iosPrj的Chapter05-
StudentPerformance4.0



编写刷新事件处理函数

■ 刷新控件的事件处理函数 refreshing

- 判断当前刷新控件的状态是否为刷新中。
- 如果正在刷新在继续执行后续的流程，设置刷新中的 attributedTitle，
- 向studentsInfo中添加一个学生记录，调用刷新控件的 endRefreshing方法结束刷新状态，重置attributedTitle为“refreshing”。
- 调用tableView的方法 reloadData（），该方法会重新载入表格视图中的数据。

```
func refreshing() {  
    if (refreshControl?.isRefreshing == true) {  
        refreshControl?.attributedTitle = NSAttributedString  
            (string: "loading...")  
        studentsInfo.addStudent()  
        refreshControl?.endRefreshing()  
        refreshControl?.attributedTitle = NSAttributedString  
            (string: "refreshing")  
        tableView.reloadData()  
    }  
}
```



谢 谢
