

Journal

Lesson 4

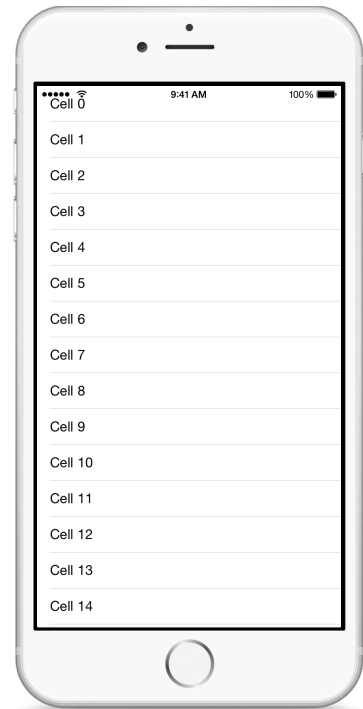


Description

Illustrate table cell reuse by adding sample data to the table view.

Learning Outcomes

- Discover how to add rows of data to a table view.
- Illustrate `UITableViewDelegate` methods and table cell reuse.
- Explore the relationship between optional types, optional binding and forced unwrapping.
- Assess different ways of declaring and initializing property values.
- Apply `map` and a closure to create a transformation of data.



Vocabulary

<code>UITableViewDataSource</code>	array	property
<code>NSIndexPath</code>	array subscripting	optional
optional binding	forced unwrapping	for-in loop
range	map	closure expression
trailing closure		

Materials

- **Journal Lesson 4** Xcode project

Opening

How can we get some data to appear in the table?

Agenda

- Begin an experiment with using a simple array of strings to illustrate how the `UITableViewDataSource` method `tableView:cellForRowAtIndexPath:` will work.
- Add a naive array property to the `JournalTableViewController` class.

```
var sampleData: [String]?
```

- Update the implementation of `viewDidLoad` to naively populate the array.

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    sampleData = [String]()  
    for var index = 0; index < 1000; ++index {  
        sampleData!.append("Cell \(index)")  
    }  
}
```

- Discuss that the `sampleData` property is an optional type, since the initializer will not assign the property its initial value; how `viewDidLoad` assigns the `sampleData` property an empty `String` array; and how a C-style for loop appends 1000 `String` values to the array, after a forced unwrapping.
- Update the implementation of `tableView:NumberOfRowsInSection:`.

```
override func tableView(tableView: UITableView,  
    numberOfRowsInSection section: Int) -> Int {  
    if let data = sampleData {  
        return data.count  
    } else {  
        return 0  
    }  
}
```

- Discuss how the `tableView:NumberOfRowsInSection:` method uses optional binding to ensure the existence of the `sampleData` array, and uses the size of the array to determine the number of rows in the table view.
- Update the implementation of `tableView:cellForRowAtIndexPath:`.

```

override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    let cell =
        tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier,
            forIndexPath: indexPath) as! UITableViewCell
    if let data = sampleData {
        if let label = cell.textLabel {
            label.text = data[indexPath.row]
        }
    }
    return cell
}

```

- Explain how optional binding is used to unwrap the `sampleData` array and the optional `textLabel` property of a table cell; and how the `indexPath.row` is used as the index for the array.
- Add a custom breakpoint to the body of the `tableView:cellForRowAtIndexPath:` method that uses a Log Message action to print obtaining reusable cell `@indexPath.row@` and automatically continue.
- Run the app (⌘R), scroll the table rows up and down, observe the different cells appear, and observe the console output.
- Explain how the table view only maintains enough `UITableViewCell` objects in memory to present the visible cells and animate the table view efficiently.
- Discuss the naive declaration of the `sampleData` property, the generation of values in the `sampleData` array, and the multiple occurrences of unwrapping.
- Improve the implementation of the `sampleData` property declaration and `viewDidLoad` with a default property value and a `for-in` loop; and remove the occurrences of unwrapping the `sampleData` property.

```

var sampleData = [String]()

override func viewDidLoad() {
    super.viewDidLoad()
    for index in 0..<1000 {
        sampleData.append("Cell \(index)")
    }
}
...
override func tableView(tableView: UITableView,
    numberOfRowsInSection: Int) -> Int {
    return sampleData.count
}

```

```
override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    let cell =
        tableView.dequeueReusableCellWithIdentifier(cellReuseIdentifier,
            forIndexPath: indexPath) as! UITableViewCell
    if let label = cell.textLabel {
        label.text = sampleData[indexPath.row]
    }
    return cell
}
```

- Discuss how the `for-in` loop is more succinct, and using a default property value for the `sampleData` property removes the need to use an optional type.
- Run the app (⌘R), and observe that the functionality remains the same.
- Discuss how `map` might be used to improve the property declaration.
- Improve the property declaration with a call of `map` and a closure expression.

```
let sampleData = (0..<1000).map( { (index: Int) -> String in
    return "Cell \(index)"
})

override func viewDidLoad() {
    super.viewDidLoad()
}
```

- Explain that the `sampleData` is now a constant with a default value generated by transforming a range of `Int` values into an array of `String` values.
- Run the app (⌘R), and observe that the functionality remains the same.
- Discuss how the closure expression can be more succinct by leveraging Swift type inference, implicit closure parameters and the trailing closure syntax.
- Replace the verbose closure expression in the `sampleData` property declaration with a succinct trailing closure expression.

```
let sampleData = (0..<1000).map { "Cell \($0)" }
```

- Run the app (⌘R), and observe that the functionality remains the same.

Closing

What happens when the cell reuse identifier string used in `tableView:cellForRowAtIndexPath:` does not match the identifier of the prototype cell specified in Interface Builder?

Modifications and Extensions

- Change the custom breakpoint to print the memory address of the `UITableViewCell` instance. Inspect the console output to confirm repeated memory addresses, and explain how this is related to cell reuse.

Resources

Table View Programming Guide for iOS https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/TableView_iPhone/AboutTableViewsiPhone/AboutTableViewsiPhone.html

Cocoa Core Competencies: Delegation <https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html>

Cocoa Core Competencies: Protocol <https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Protocol.html>

UIKit User Interface Catalog: Table Views <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/UIKitUICatalog/UITableView.html>

UITableViewController Class Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewController_Class/index.html

UITableView Class Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableView_Class/index.html

UITableViewDelegate Protocol Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDelegate_Protocol/index.html

UITableViewDataSource Protocol Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewDataSource_Protocol/index.html

NSIndexPath Class Reference https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSIndexPath_Class/index.html

The Swift Programming Language: Control Flow https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ControlFlow.html

The Swift Programming Language: Closures https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Closures.html