Taming UITableViews and their content

Copenhagen Cocoa May 2016

Marius Constantinescu
@marius_const

@nodes_ios

What’s wrong with `UITableView`?

A lot of boilerplate code

- copy-paste methods for many UIViewControllers
- a lot of methods to implement

UITableView if-soup

```
if indexPath.section == 0 {
    if indexPath.row == 0 {

    } else if indexPath.row == 1 {

    } else {

    }
} else if indexPath.section == 1 {

} else {

}
```

UITableViewController

- very specific
- various bugs
- hard to customize (insets, toolbar, background image view, etc)

- Protocols and protocol oriented programming
- Default implementations
- Protocol composition
- `typealias TableController = protocol<UITableViewDataSource, UITableViewDelegate>`

Extensions

- ability to add functions to existing classes/structs/protocols

Generics

- Less Code
- Makes you think in a bigger context
- Enforces consistency if paired with protocols
- Lots of bugs in Swift compiler
- Segmentation fault: 11

Sourcery

- Suits 90% of screens with table views [1]
- Generic data source & delegate
- Takes DataType and CellType
- Closures for population, selection handling, etc

[1] Educated guess

Protocols

```
public protocol TableViewPresentable: NibInstantiable {
    static var nib: UINib { get }
    static var reuseIdentifier: String { get }
    static var staticHeight: CGFloat { get }
    static var loadsFromNib: Bool { get }
}

public extension TableViewPresentable {
    public static var nib: UINib {
        return UINib(nibName: String(self), bundle: nil)
    }

    public static func newFromNib<T>() -> T {
        return nib.instantiateWithOwner(nil, options: nil).first as! T
    }

    public static var reuseIdentifier: String {
        return String(self)
    }

    public static var loadsFromNib: Bool {
        return true
    }
}
```

Data Sources

SimpleSourcery

- One line setup
- Easy to replace, manipulate or change

```
sourcery = SimpleSourcery<String, BasicCell>(tableView: tableView, data: data, configurator: { $0.cell.textLabel?.text = $0.object })
```

ComplexSourcery

- Based on `Section`
- Each `Section` is similar to a `SimpleSourcery`
- Different DataType and CellType in different Sections
- Support for headers
- Easy to setup

```
let textSection = Section<String, BasicCell>(
                                  title: nil,
                                   data: data.texts,
                              configurator: { $0.cell.textLabel?.text = $0.object },
                         selectionHandler: nil)

let colorSection = Section<UIColor, ColorCell>(
                                  title: "Colors",
                                   data: data.colors,
                              configurator: { $0.cell.populateWithColor($0.object) },
                          selectionHandler: nil)

sourcery = ComplexSourcery(tableView: tableView, sections: [textSection, colorSection])
```

PagedSourcery

- Ideal for search, comments, feeds or anything that's paged
- Provides an NSOperationQueue
- Shows all rows, loads them on demand
- Requirement: the API must return the total count
- Ability to preload

Demo

Project at https://github.com/mariusc/SourceryDemo

Resources

- https://github.com/nodes-ios/Sourcery
- http://www.iosnomad.com/blog/2014/4/21/fluent-pagination
- http://www.brewerydb.com/