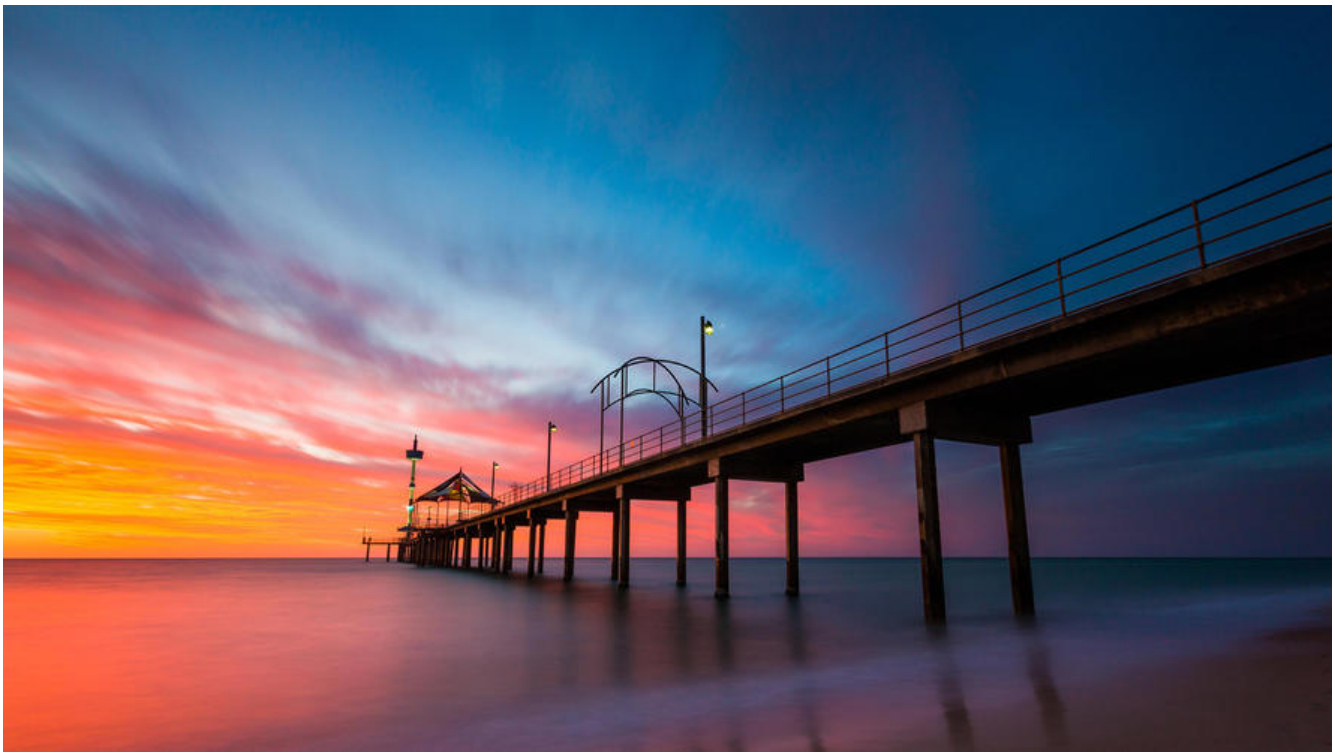
UITableView 和 UICollectionView 的新渲染方式 DiffableDataSource

作者：NilPixel

原文链接：https://ld246.com/article/1600746538794

来源网站：链滴



背景（current state of the art)

iOS13 之前，在 UIKit 中 UITableView 和 UICollectionView 和数据源进行交互时一定会用到如下三协议方法：

- UITableView

```
@available(iOS 2.0, *)
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int


// Row display. Implementers should *always* try to reuse cells by setting each cell's reuseIde
tifier and querying for available reusable cells with dequeueReusableCellWithIdentifier:
// Cell gets various attributes set automatically based on table (separators) and data source (a
cessory views, editing controls)

@available(iOS 2.0, *)
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableView
ell


@available(iOS 2.0, *)
optional func numberOfSections(in tableView: UITableView) -> Int // Default is 1 if not imple
ented
```

- UICollectionView

```
@available(iOS 6.0, *)
func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int)
> Int
```

原文链接：UITableView 和 UICollectionView 的新渲染方式 DiffableDataSource

```
// The cell that is returned must be retrieved from a call to -dequeueReusableCellWithReuseI
entifier:forIndexPath:
@available(iOS 6.0, *)
func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) ->
UICollectionViewCell


@available(iOS 6.0, *)
optional func numberOfSections(in collectionView: UICollectionView) -> Int
```

在渲染 UITableView 和 UICollectionView 时，这几个方法会实时的根据 indexPath 为我们返回对的数据，从而渲染出我们需要的对应的 cell。这很简单，接触过 iOS 开发的同学们都知道的，这几个法已经服务了我们将近 10 年了，这 10 年间我们早已经习惯了它们，习惯了它们的简单直接，同时一直忍受着它们带来的各种莫名其妙的问题。比如：某些时候当你调用 performBatchUpdates()时

```
'NSInternalInconsistencyException', reason: 'Invalid update: invalid number of rows in section
. The number of rows contained in an existing section after the update (0) must be equal to t
e number of rows contained in that section before the update (3), plus or minus the number o
 rows inserted or deleted from that section (0 inserted, 1 deleted) and plus or minus the num
er of rows moved into or out of that section (0 moved in, 0 moved out).'
```

当 UI 和数据源之间的同步出现问题时，这种奇怪的事情就会发生。如果这时候你换成调用 reloadDat()，这个问题可以被暂时解决，但是 UI 状态的切换没有动画，过渡生硬，和优秀的交互理念背道而驰。

所以，我们看到，上面我们平日里熟练使用的给 UITableView 和 UICollectionView 提供数据源的方虽然简单，但是在使用的时候很容易出错。

新方法（a new approach）

DiffableDataSource，苹果官方演示视频里把它翻译成“差量数据源”。

从此没有了 performBatchUpdates(),只有简单的 apply()方法。

在此先引入一个新概念 snapshot，快照。快照可以理解成 APP 在运行过程中的某一个时刻的某一个态。这里由于加入了唯一标识符，每个 section 和 item 的状态都是唯一的，也就是说 UITableView UICollectionView 的在渲染过程中，每一个状态都是唯一的。所以，当 section 或 item 里的数据变动时，DiffableDataSource 可以很轻松的检测到，并将变动后的数据源和变动前的数据源做对比得出差异，渲染到界面上。DiffableDataSource 就是调用了一个简单的 apply()方法，完成了旧快照新快照的渲染过程。所以，DiffableDataSource 的核心是唯一标识符，从此就再也没有了 IndexPat，一切都是 identifier。

这里涉及到四个类： UITableViewDiffableDataSource 、 UICollectionViewDiffableDataSource 、SCollectionViewDiffableDataSource 、 NSDiffableDataSourceSnapshot 。在 iOS 平台上是 UITaleViewDiffableDataSource 和 UICollectionViewDiffableDataSource ， Mac 平台上是 NSCollectinViewDiffableDataSource 。 NSDiffableDataSourceSnapshot 是所有平台通用的。

新方法实践（practice）

分三步，如下面代码所示：

1. 配置 UI
2. 配置数据源
3. 数据源应用快照，更新 UI

```
override func viewDidLoad() {
    super.viewDidLoad()
    configureTableView()    // 配置UI
    configDataSource()      // 配置数据源
    updateUI()              // 数据源应用快照，更新UI
}
```

配置 UI 相关代码：

```
func configureTableView() {
    view.addSubview(tableView)
    tableView.translatesAutoresizingMaskIntoConstraints = false
    NSLayoutConstraint.activate([
        tableView.leadingAnchor.constraint(equalTo: view.leadingAnchor),
        tableView.trailingAnchor.constraint(equalTo: view.trailingAnchor),
        tableView.topAnchor.constraint(equalTo: view.topAnchor),
        tableView.bottomAnchor.constraint(equalTo: view.bottomAnchor)
        ])
    tableView.register(UITableViewCell.self, forCellReuseIdentifier: ViewController.reuseIdenti
ier)
}
```

很简单，就是在 ViewController 中添加一个 tableView，添加好约束，然后注册一下 cell。

配置数据源

```
func configDataSource() {
    self.dataSource = UITableViewDiffableDataSource<Section, Item>.init(tableView: self.tab
eView, cellProvider: {(tableView, indexPath, item) -> UITableViewCell? in
        let cell = tableView.dequeueReusableCell(withIdentifier: ViewController.reuseIdentifier,
for: indexPath)
        var content = cell.defaultContentConfiguration()
        content.text = item.title
        cell.contentConfiguration = content
        return cell
    })
    self.dataSource.defaultRowAnimation = .fade
}
```

这里就涉及到本文的主角了 UITableViewDiffableDataSource ，创建一个 UITableViewDiffableDat Source 的实例，注意，这里初始化方法里需要把当前的 tableView 传进去，是谁的 dataSource 就谁进去。 UITableViewDiffableDataSource 可以用 section 和 item 的泛型约束一下。然后，在 cell rovider 这个 closure 回调里进行 cell 的赋值等内容相关配置。

数据源应用快照，更新 UI

```
func updateUI() {
    currentSnapshot = NSDiffableDataSourceSnapshot<Section, Item>()
    currentSnapshot.appendSections([.main])
    currentSnapshot.appendItems(mainItems, toSection: .main)
    self.dataSource.apply(currentSnapshot, animatingDifferences: true)
}
```

这里要创建一个快照 NSDiffableDataSourceSnapshot 的实例，在这个快照实例里面添加 section 据，并为对应的 section 添加对应的 item 数据。需要说明一下的是，Apple 把这里需要传进快照里数据抽象成标识符，也就是传进去的是标识符数据，需要满足数据的唯一性，所以，传进去的数据需满足可 hash 这个条件。代码里 sections 传入的是枚举数组，swift 的枚举是自动可 hash 的，items 传进去的是自定义的 struct，实现了 Hashable 协议，如下所示：

```
enum Section: CaseIterable {
    case main
}

struct Item: Hashable {
    let title: String

    init(title: String) {
        self.title = title
        self.identifier = UUID()
    }

    private let identifier: UUID

    func hash(into hasher: inout Hasher) {
        hasher.combine(self.identifier)
    }
}
```

好了，至此，组建好数据结构：

```
    lazy var mainItems: [Item] = {
        return [Item(title: "标题1"),
                Item(title: "标题2"),
                Item(title: "标题3")]
    }()
```

将其传入当前的快照实例 currentSnapshot，最后数据源应用一下这个快照 self.dataSource.apply(currentSnapshot, animatingDifferences: true) 列表就渲染成功了，还自带优秀动画哦。

demo地址