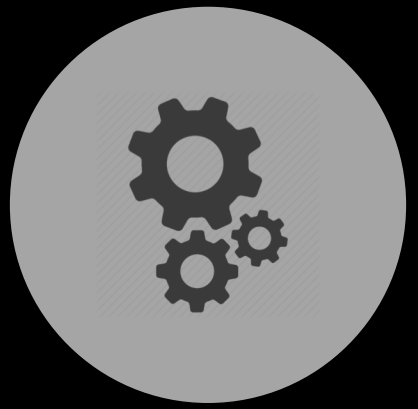
Developing an app with DiffableDataSource

Jeroen Bakker

@JeroenBiOS

DataSources

PROVIDING DATA



CONFIGURES CELLS

NUMBER OF ITEMS/SECTIONS

Providing data

```
9   import Foundation
10
11  struct Speaker {
12      let name: String
13      let twitterHandler: String
14      private(set) var imageURL: URL?
15  }
```

Providing data

```swift
import Foundation

struct Speaker {
    let name: String
    let twitterHandler: String
    private(set) var imageURL: URL?
}
```

```swift

_ = Speaker(name: "Jeroen", twitterHandler: "@JeroenBiOS", imageURL: URL(string:
    "https://pragmaconference.com/assets/images/speakers/jeroen_bakker.jpg"))

_ = Speaker(name: "Jeroen", twitterHandler: "@JeroenBiOS")

```

Providing data

```
9   import UIKit
10
11  final class SpeakerCollectionViewCell: UICollectionViewCell {
12
◉       @IBOutlet weak private var avatarImageView: UIImageView!
◉       @IBOutlet weak private var titleLabel: UILabel!
◉       @IBOutlet weak private var twitterHandler: UILabel!
16
17  }
```

Providing data

```
38  // MARK: - Display
39  extension SpeakerCollectionViewCell {
40
41      func display(viewModel: Speaker) {
42          titleLabel.text = viewModel.name
43          twitterHandler.text = viewModel.twitterHandler
44
45          guard let imageURL = viewModel.imageURL else { return }
46          Nuke.loadImage(with: imageURL, into: avatarImageView)
47      }
48  }
```

Providing data

```
9   import UIKit
10
11  final class ViewController: UIViewController {
12
13      @IBOutlet weak private var collectionView: UICollectionView!
14      private var speakers: [Speaker] = []
15  }
```

Configure cells

```
24  // MARK: - Setup
25  extension ViewController {
26
27      func setupCollectionView() {
28          collectionView.register(UINib(nibName: String(describing:
                SpeakerCollectionViewCell.self), bundle: nil), forCellWithReuseIdentifier:
                String(describing: SpeakerCollectionViewCell.self))
29
30          collectionView.delegate = self
31          collectionView.dataSource = self
32      }
33  }
```

Providing data & Configure cells

```
46  // MARK: - UICollectionViewDataSource
47  extension ViewController: UICollectionViewDataSource {
48
49      func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
            IndexPath) -> UICollectionViewCell {
50          guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
                String(describing: SpeakerCollectionViewCell.self), for: indexPath) as?
                SpeakerCollectionViewCell else {
51              fatalError("Could not dequeue SpeakerCollectionViewCell")
52          }
53
54          cell.display(viewModel: speakers[indexPath.item])
55
56          return cell
57      }
58  }
```

Number of items/sections

```
46  // MARK: - UICollectionViewDataSource
47  extension ViewController: UICollectionViewDataSource {
48
49      func numberOfSections(in collectionView: UICollectionView) -> Int {
50          return 1
51      }
52
53      func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section:
            Int) -> Int {
54          return speakers.count
55      }
```

Layout

```
68  // MARK: - UICollectionViewDelegateFlowLayout
69  extension ViewController: UICollectionViewDelegateFlowLayout {
70
71      func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
            UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
72          return CGSize(width: collectionView.frame.width, height: 62)
73      }
74  }
```

Dummy

```
35  extension ViewController {
36
37      func populateCollectionView() {
38          for i in 1..<100 {
39              speakers.append(Speaker(
40                  name: "User \(i)",
41                  twitterHandler: "@Twitter \(i)",
42                  imageURL: URL(string:
                        "https://pragmaconference.com/assets/images/speakers/jeroen_bakker.jpg")
43              ))
44          }
45
46          collectionView.reloadData()
47      }
48  }
```

Dummy


21:12
User 1
@Twitter 1
User 2
@Twitter 2
User 3
@Twitter 3
User 4
@Twitter 4
User 5
@Twitter 5
User 6
@Twitter 6
User 7
@Twitter 7
User 8
@Twitter 8
User 9
@Twitter 9
User 10
@Twitter 10
User 11
@Twitter 11
User 12
@Twitter 12

Animations

- 10 item > 1 items
- 1 items > 10 item
- 5 items > 5 new items

Difference

- 10 item > 1 items

- We have to reload item 1
- We have to delete item 2..<10

Difference

- 1 item > 10 items

- We have to reload item 1
- We have to insert item 2..<10

Difference

- 5 item > 5 items

- We have to reload all items
- We do not want to use reloadData though

PerformBatchUpdates

```
63      func populateCollectionView(with speakers: [Speaker]) {
64          let previousSpeakers = self.speakers
65          self.speakers = speakers
66
67          collectionView.performBatchUpdates({
68              if previousSpeakers.count > speakers.count {
69                  let reloadIndexPaths = Array(0..<speakers.count).map({ IndexPath(item: $0, section: 0) })
70                  let deleteIndexPaths = Array(speakers.count..<previousSpeakers.count).map({ IndexPath(item: $0, section: 0) })
71                  collectionView.reloadItems(at: reloadIndexPaths)
72                  collectionView.deleteItems(at: deleteIndexPaths)
73              } else if previousSpeakers.count < speakers.count {
74                  let reloadIndexPaths = Array(0..<previousSpeakers.count).map({ IndexPath(item: $0, section: 0) })
75                  let insertIndexPaths = Array(previousSpeakers.count..<speakers.count).map({ IndexPath(item: $0, section: 0) })
76                  collectionView.reloadItems(at: reloadIndexPaths)
77                  collectionView.insertItems(at: insertIndexPaths)
78              } else {
79                 let indexPaths = Array(0..<speakers.count).map({ IndexPath(item: $0, section: 0) })
80                 collectionView.reloadItems(at: indexPaths)
81              }
82          }, completion: nil)
83      }
84  }
```

PerformBatchUpdates

```
*** Terminating app due to uncaught
exception
'NSInternalInconsistencyException',
reason: 'attempt to delete item * from
section * which only contains * items
before the update'
```

https://www.pinclipart.com/pindetail/hhmxwx_anger-disney-wiki-cartoon-and-pixar-inside-inside/

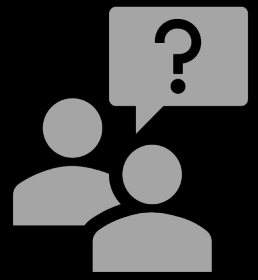
WWDC 19

Deprecated:

performbatchUpdates

insertItems/deleteItems deprecated

No more indexPaths, instead identifiers



New: DiffableDataSource

Single source of truth

Use the method “apply()” for updates

DiffableDataSource

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {
```

DiffableDataSource

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {
334
```

DiffableDataSource

```
332 @available(iOS 13.0, tvOS 13.0, *)
333 open class UICollectionViewDiffableDataSource<SectionIdentifierType,
        ItemIdentifierType> : NSObject, UICollectionViewDataSource where
        SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {
```

DiffableDataSource

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {
```

DiffableDataSource

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


341      public init(collectionView: UICollectionView, cellProvider: @escaping
             UICollectionViewDiffableDataSource<SectionIdentifierType,
             ItemIdentifierType>.CellProvider)
```

DiffableDataSource

```
332 @available(iOS 13.0, tvOS 13.0, *)
333 open class UICollectionViewDiffableDataSource<SectionIdentifierType,
        ItemIdentifierType> : NSObject, UICollectionViewDataSource where
        SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {
334
341     public init(collectionView: UICollectionView, cellProvider: @escaping
            UICollectionViewDiffableDataSource<SectionIdentifierType,
            ItemIdentifierType>.CellProvider)
```

DiffableDataSource

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


341      public init(collectionView: UICollectionView, cellProvider: @escaping
             UICollectionViewDiffableDataSource<SectionIdentifierType,
             ItemIdentifierType>.CellProvider)
```

DiffableDataSource

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


341      public init(collectionView: UICollectionView, cellProvider: @escaping
             UICollectionViewDiffableDataSource<SectionIdentifierType,
             ItemIdentifierType>.CellProvider)
```

DiffableDataSource

```
332 @available(iOS 13.0, tvOS 13.0, *)
333 open class UICollectionViewDiffableDataSource<SectionIdentifierType,
        ItemIdentifierType> : NSObject, UICollectionViewDataSource where
        SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


341     public init(collectionView: UICollectionView, cellProvider: @escaping
            UICollectionViewDiffableDataSource<SectionIdentifierType,
            ItemIdentifierType>.CellProvider)


335     public typealias CellProvider = (UICollectionView, IndexPath,
            ItemIdentifierType) -> UICollectionViewCell?
```

DiffableDataSource

```
332 @available(iOS 13.0, tvOS 13.0, *)
333 open class UICollectionViewDiffableDataSource<SectionIdentifierType,
        ItemIdentifierType> : NSObject, UICollectionViewDataSource where
        SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


341     public init(collectionView: UICollectionView, cellProvider: @escaping
            UICollectionViewDiffableDataSource<SectionIdentifierType,
            ItemIdentifierType>.CellProvider)


335     public typealias CellProvider = (UICollectionView, IndexPath,
            ItemIdentifierType) -> UICollectionViewCell?
```

Providing data

```
9   import Foundation
10
11  struct Speaker {
12      let name: String
13      let twitterHandler: String
14      private(set) var imageURL: URL?
15  }
```

Providing data

```swift
import Foundation

struct Speaker {
    private let identifier: UUID = UUID()

    let name: String
    let twitterHandler: String
    private(set) var imageURL: URL?
}

extension Speaker: Hashable {

    // Hashable
    func hash(into hasher: inout Hasher) {
        hasher.combine(identifier)
    }

    // Equatable - Inherited from Hashable
    static func == (lhs: Speaker, rhs: Speaker) -> Bool {
        return lhs.identifier == rhs.identifier
    }
}
```

Providing data

```
9   import UIKit
10
11  final class ViewController: UIViewController {
12
13      @IBOutlet weak private var collectionView: UICollectionView!
14      private var speakers: [Speaker] = []
15  }
```

Providing data

```
9   import UIKit
10
11  final class ViewController: UIViewController {
12
13      enum Section {
14          case main
15      }
16
17      @available(iOS 13.0, *)
18      private lazy var dataSource: UICollectionViewDiffableDataSource<Section, Speaker>? = nil
19
20      @IBOutlet weak private var collectionView: UICollectionView!
21      private var speakers: [Speaker] = []
22  }
```

Providing data

```
34      @available(iOS 13.0, *)
35      func setupCollectionViewDataSource() {
36          dataSource = .init(collectionView: collectionView, cellProvider: { (collectionView,
                indexPath, speaker) -> UICollectionViewCell? in
37              guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
                    String(describing: SpeakerCollectionViewCell.self), for: indexPath) as?
                    SpeakerCollectionViewCell else {
38                  fatalError("Could not dequeue SpeakerCollectionViewCell")
39              }
40
41              cell.display(viewModel: speaker)
42
43              return cell
44          })
45      }
```

Providing data

```
34      @available(iOS 13.0, *)
35      func setupCollectionViewDataSource() {
36          dataSource = .init(collectionView: collectionView, cellProvider: { (collectionView,
                indexPath, speaker) -> UICollectionViewCell? in
37              guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
                    String(describing: SpeakerCollectionViewCell.self), for: indexPath) as?
                    SpeakerCollectionViewCell else {
38                  fatalError("Could not dequeue SpeakerCollectionViewCell")
39              }
40
41              cell.display(viewModel: speaker)
42
43              return cell
44          })
45      }
```

Providing data

```
34      @available(iOS 13.0, *)
35      func setupCollectionViewDataSource() {
36          dataSource = .init(collectionView: collectionView, cellProvider: { (collectionView,
                indexPath, speaker) -> UICollectionViewCell? in
37              guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
                    String(describing: SpeakerCollectionViewCell.self), for: indexPath) as?
                    SpeakerCollectionViewCell else {
38                  fatalError("Could not dequeue SpeakerCollectionViewCell")
39              }
40
41              cell.display(viewModel: speaker)
42
43              return cell
44          })
45      }
```

Providing data

```
34      @available(iOS 13.0, *)
35      func setupCollectionViewDataSource() {
36          dataSource = .init(collectionView: collectionView, cellProvider: { (collectionView,
                indexPath, speaker) -> UICollectionViewCell? in
37              guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
                    String(describing: SpeakerCollectionViewCell.self), for: indexPath) as?
                    SpeakerCollectionViewCell else {
38                  fatalError("Could not dequeue SpeakerCollectionViewCell")
39              }
40
41              cell.display(viewModel: speaker)
42
43              return cell
44          })
45      }
```

Providing data

```
96      func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
            IndexPath) -> UICollectionViewCell {
97          guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
                String(describing: SpeakerCollectionViewCell.self), for: indexPath) as?
                SpeakerCollectionViewCell else {
98              fatalError("Could not dequeue SpeakerCollectionViewCell")
99          }
100
101         cell.display(viewModel: speakers[indexPath.item])
102
103         return cell
104     }
```

Providing data

```
34      @available(iOS 13.0, *)
35      func setupCollectionViewDataSource() {
36          dataSource = .init(collectionView: collectionView, cellProvider: { (collectionView,
                indexPath, speaker) -> UICollectionViewCell? in
37              guard let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
                    String(describing: SpeakerCollectionViewCell.self), for: indexPath) as?
                    SpeakerCollectionViewCell else {
38                  fatalError("Could not dequeue SpeakerCollectionViewCell")
39              }
40
41              cell.display(viewModel: speaker)
42
43              return cell
44          })
45      }
```

Configure cells

```
24  // MARK: - Setup
25  extension ViewController {
26
27      func setupCollectionView() {
28          collectionView.register(UINib(nibName: String(describing:
                SpeakerCollectionViewCell.self), bundle: nil), forCellWithReuseIdentifier:
                String(describing: SpeakerCollectionViewCell.self))
29
30          collectionView.delegate = self
31          collectionView.dataSource = self
32      }
33  }
```

Configure cells

```
24  // MARK: - Setup
25  extension ViewController {
26
27      func setupCollectionView() {
28          collectionView.register(UINib(nibName: String(describing:
                SpeakerCollectionViewCell.self), bundle: nil), forCellWithReuseIdentifier:
                String(describing: SpeakerCollectionViewCell.self))
29
30          collectionView.delegate = self
31          if #available(iOS 13.0, *) {
32              setupCollectionViewDataSource()
33          } else {
34              collectionView.dataSource = self
35          }
36      }
```

What have we done so far

- Updated our Speaker model to conform to Hashable
- Created a Section enum with 1 case “main”
- Added diffableDatasource property
- Configured the dataSource

How to populate?

- Snapshot
- Snapshot of our current dataSource state
- Fresh clean snapshot
- Apply the snapshot to the dataSource

How to populate?

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


343      open func apply(_ snapshot:
             NSDiffableDataSourceSnapshot<SectionIdentifierType,
             ItemIdentifierType>, animatingDifferences: Bool = true, completion: (()
             -> Void)? = nil)
```

How to populate?

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {
334
343      open func apply(_ snapshot:
             NSDiffableDataSourceSnapshot<SectionIdentifierType,
             ItemIdentifierType>, animatingDifferences: Bool = true, completion: (()
             -> Void)? = nil)
```

How to populate?

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


343      open func apply(_ snapshot:
             NSDiffableDataSourceSnapshot<SectionIdentifierType,
             ItemIdentifierType>, animatingDifferences: Bool = true, completion: (()
             -> Void)? = nil)
```

How to populate?

```
35  extension ViewController {
36
37      func populateCollectionView() {
38          for i in 1..<100 {
39              speakers.append(Speaker(
40                  name: "User \(i)",
41                  twitterHandler: "@Twitter \(i)",
42                  imageURL: URL(string:
                        "https://pragmaconference.com/assets/images/speakers/jeroen_bakker.jpg")
43              ))
44          }
45
46          collectionView.reloadData()
47      }
48  }
```

How to populate?

```
54      func populateCollectionView() {
55          for i in 1..<100 {•••}
62
63          if #available(iOS 13.0, *) {
64              var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
65              snapshot.appendSections([Section.main])
66              snapshot.appendItems(speakers)
67              dataSource?.apply(snapshot)
68          } else {
69              collectionView.reloadData()
70          }
71      }
```

Layout

```
75      func populateCollectionView() {
76          for i in 1..<100 {
77              speakers.append(Speaker(
78                  name: "User \(i)",
79                  twitterHandler: "@Twitter \(i)",
80                  imageURL: URL(string:
                        "https://pragmaconference.com/assets/images/speakers/jeroen_bakker.jpg")
81              ))
82              cellSizes.append(CGSize(width: view.frame.width - 32, height: 62))
83          }
```

Layout

```
149     func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
            UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
150         return cellSizes[indexPath.item]
151     }
```

Layout

```
22      private var speakers: [Speaker] = []
23      private var cellSizes: [CGSize] = []
24      private lazy var newCellSizes: [Int: CGSize] = [:]
```

Layout

```
76      func populateCollectionView() {
77          for i in 1..<100 {
78              let speaker = Speaker(
79                  name: "User \(i)",
80                  twitterHandler: "@Twitter \(i)",
81                  imageURL: URL(string:
                        "https://pragmaconference.com/assets/images/speakers/jeroen_bakker.jpg")
82              )
83              speakers.append(speaker)
84              if #available(iOS 13.0, *) {
85                  newCellSizes[speaker.hashValue] = CGSize(width: view.frame.width - 32, height: 62)
86              } else {
87                  cellSizes.append(CGSize(width: view.frame.width - 32, height: 62))
88              }
89          }
```

How do we get the speaker?

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


347      open func itemIdentifier(for indexPath: IndexPath) -> ItemIdentifierType?
348
349      open func indexPath(for itemIdentifier: ItemIdentifierType) -> IndexPath?
```

Layout

```
155     func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
            UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
156         if #available(iOS 13.0, *), let identifier = dataSource?.itemIdentifier(for: indexPath) {
157             return newCellSizes[identifier.hashValue] ?? .zero
158         } else {
159             return cellSizes[indexPath.item]
160         }
161     }
```

Dummy


21:12
User 1
@Twitter 1
User 2
@Twitter 2
User 3
@Twitter 3
User 4
@Twitter 4
User 5
@Twitter 5
User 6
@Twitter 6
User 7
@Twitter 7
User 8
@Twitter 8
User 9
@Twitter 9
User 10
@Twitter 10
User 11
@Twitter 11
User 12
@Twitter 12

PerformBatchUpdates

```
63      func populateCollectionView(with speakers: [Speaker]) {
64          let previousSpeakers = self.speakers
65          self.speakers = speakers
66
67          collectionView.performBatchUpdates({
68              if previousSpeakers.count > speakers.count {
69                  let reloadIndexPaths = Array(0..<speakers.count).map({ IndexPath(item: $0, section: 0) })
70                  let deleteIndexPaths = Array(speakers.count..<previousSpeakers.count).map({ IndexPath(item: $0, section: 0) })
71                  collectionView.reloadItems(at: reloadIndexPaths)
72                  collectionView.deleteItems(at: deleteIndexPaths)
73              } else if previousSpeakers.count < speakers.count {
74                  let reloadIndexPaths = Array(0..<previousSpeakers.count).map({ IndexPath(item: $0, section: 0) })
75                  let insertIndexPaths = Array(previousSpeakers.count..<speakers.count).map({ IndexPath(item: $0, section: 0) })
76                  collectionView.reloadItems(at: reloadIndexPaths)
77                  collectionView.insertItems(at: insertIndexPaths)
78              } else {
79                 let indexPaths = Array(0..<speakers.count).map({ IndexPath(item: $0, section: 0) })
80                 collectionView.reloadItems(at: indexPaths)
81              }
82          }, completion: nil)
83      }
84  }
```

Apply

```
92      func populateCollectionView(with speakers: [Speaker]) {
93          let previousSpeakers = self.speakers
94          self.speakers = speakers
95
96          if #available(iOS 13.0, *), var snapshot = dataSource?.snapshot() {
97              snapshot.deleteItems(previousSpeakers)
98              snapshot.appendItems(speakers)
99              dataSource?.apply(snapshot, animatingDifferences: true)
100         } else {
101             collectionView.performBatchUpdates( ••• )
117         }
118     }
```

Apply

```
92      func populateCollectionView(with speakers: [Speaker]) {
93          let previousSpeakers = self.speakers
94          self.speakers = speakers
95
96          if #available(iOS 13.0, *), var snapshot = dataSou       apshot() {
97              snapshot.deleteItems(previousSpeakers)
98              snapshot.appendItems(speakers)
99              dataSource?.apply(snapshot, animatingDifferen     : true)
100         } else {
101             collectionView.performBatchUpdates( ••• )
117         }
118     }
```

https://www.stickpng.com/img/at-the-movies/cartoons/inside-out/joy-happy

Snapshot

```
298     public mutating func appendItems(_ identifiers: [ItemIdentifierType], toSection sectionIdentifier:
            SectionIdentifierType? = nil)
299
300     public mutating func insertItems(_ identifiers: [ItemIdentifierType], beforeItem beforeIdentifier:
            ItemIdentifierType)
301
302     public mutating func insertItems(_ identifiers: [ItemIdentifierType], afterItem afterIdentifier:
            ItemIdentifierType)
303
304     public mutating func deleteItems(_ identifiers: [ItemIdentifierType])
305
306     public mutating func deleteAllItems()
307
308     public mutating func moveItem(_ identifier: ItemIdentifierType, beforeItem toIdentifier:
            ItemIdentifierType)
309
310     public mutating func moveItem(_ identifier: ItemIdentifierType, afterItem toIdentifier:
            ItemIdentifierType)
311
312     public mutating func reloadItems(_ identifiers: [ItemIdentifierType])
```

Snapshot

```
314     public mutating func appendSections(_ identifiers: [SectionIdentifierType])
315
316     public mutating func insertSections(_ identifiers: [SectionIdentifierType], beforeSection
            toIdentifier: SectionIdentifierType)
317
318     public mutating func insertSections(_ identifiers: [SectionIdentifierType], afterSection
            toIdentifier: SectionIdentifierType)
319
320     public mutating func deleteSections(_ identifiers: [SectionIdentifierType])
321
322     public mutating func moveSection(_ identifier: SectionIdentifierType, beforeSection toIdentifier:
            SectionIdentifierType)
323
324     public mutating func moveSection(_ identifier: SectionIdentifierType, afterSection toIdentifier:
            SectionIdentifierType)
325
326     public mutating func reloadSections(_ identifiers: [SectionIdentifierType])
```

Snapshot Keep in mind

- Order is very important
- Uniqueness is very important
- Apply from background threads

Order is very important

```
73      func populateCollectionView() {
74          for i in 1..<10 {•••}
81
82          if #available(iOS 13.0, *) {
83              var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
84              snapshot.appendItems(speakers)
85              snapshot.appendSections([Section.main])
86              dataSource?.apply(snapshot)
87          } else {
88              collectionView.reloadData()
89          }
90      }
```

Order is very important

```
*** Terminating app due to uncaught
exception
'NSInternalInconsistencyException',
reason: 'There are currently no sections
in the data source. Please add a section
first.'
```

Order is very important

```
74      func populateCollectionView() {
75          for i in 1..<10 {•••}
82
83          if #available(iOS 13.0, *) {
84              var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
85              snapshot.appendSections([Section.main, Section.second])
86              snapshot.appendItems(speakers)
87              dataSource?.apply(snapshot)
88          } else {
89              collectionView.reloadData()
90          }
91      }
```

Order is very important

```
74      func populateCollectionView() {
75          for i in 1..<10 {•••}
82
83          if #available(iOS 13.0, *) {
84              var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
85              snapshot.appendSections([Section.main, Section.second])
86              snapshot.appendItems(speakers, toSection: Section.main)
87              dataSource?.apply(snapshot)
88          } else {
89              collectionView.reloadData()
90          }
91      }
```

Uniqueness is very important

```
74      func populateCollectionView() {
75          for i in 1..<10 {•••}
82
83          if #available(iOS 13.0, *) {
84              var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
85              snapshot.appendSections([Section.main, Section.main])
86              snapshot.appendItems(speakers, toSection: Section.main)
87              dataSource?.apply(snapshot)
88          } else {
89              collectionView.reloadData()
90          }
91      }
```

Uniqueness is very important

```
*** Terminating app due to uncaught
exception
'NSInternalInconsistencyException',
reason: 'Fatal: supplied identifiers are
not unique.'
```

Uniqueness is very important

```
74      func populateCollectionView() {
75          for i in 1..<10 {•••}
82
83          if #available(iOS 13.0, *) {
84              var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
85              snapshot.appendSections([Section.main, Section.second])
86              snapshot.appendItems(speakers, toSection: Section.main)
87              snapshot.appendItems(speakers, toSection: Section.second)
88              dataSource?.apply(snapshot)
89          } else {
90              collectionView.reloadData()
91          }
92      }
```

Uniqueness is very important

```
[UIDiffableDataSource] Warning: 9 inserted
identifier(s) already present; existing
items will be moved into place for this
current insertion. Please note this will
impact performance if items are not unique
when inserted.
```

Uniqueness is very important

```
88      func populateCollectionView() {
89          for i in 1..<100 {•••}
102
103         if #available(iOS 13.0, *) {
104             var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
105             snapshot.appendSections([Section.main])
106             snapshot.appendItems([speakers[0], speakers[0]], toSection: Section.main)
107             self.dataSource?.apply(snapshot)
108         } else {
109             collectionView.reloadData()
110         }
111     }
```

Uniqueness is very important

```
*** Terminating app due to uncaught
exception
'NSInternalInconsistencyException',
reason: 'Fatal: supplied identifiers are
not unique.'
```

Uniqueness is very important

```
11  struct Speaker {
12      private let identifier: UUID = UUID()
13
14      let name: String
15      let twitterHandler: String
16      private(set) var imageURL: URL?
17  }
18
19  extension Speaker: Hashable {
20
21      // Hashable
22      func hash(into hasher: inout Hasher) {
23          hasher.combine(identifier)
24      }
25
26      // Equatable - Inherited from Hashable
27      static func == (lhs: Speaker, rhs: Speaker) -> Bool {
28          return lhs.identifier == rhs.identifier
29      }
30  }
```

Uniqueness is very important

```
11  struct Speaker {
12      private let identifier: UUID = UUID()
13
14      let name: String
15      let twitterHandler: String
16      private(set) var imageURL: URL?
17  }
18
19  extension Speaker: Hashable {
20
21      // Hashable
22      func hash(into hasher: inout Hasher) {
23          hasher.combine(identifier)
24      }
25
26      // Equatable - Inherited from Hashable
27      static func == (lhs: Speaker, rhs: Speaker) -> Bool {
28          return lhs.identifier == rhs.identifier
29      }
30  }
```

Uniqueness is very important

```
11  struct Speaker {
12      private let identifier: UUID = UUID()
13
14      let name: String
15      let twitterHandler: String
16      private(set) var imageURL: URL?
17  }
18
19  extension Speaker: Hashable {
20
21      // Hashable
22      func hash(into hasher: inout Hasher) {
23          hasher.combine(identifier)
24      }
25
26      // Equatable - Inherited from Hashable
27      static func == (lhs: Speaker, rhs: Speaker) -> Bool {
28          return lhs.identifier == rhs.identifier
29      }
30  }
```

Apply from background threads

```
74      func populateCollectionView() {
75          for i in 1..<10 {•••}
82
83          if #available(iOS 13.0, *) {
84              DispatchQueue(label: "backgroundqueue", qos: .background).async {
85                  var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
86                  snapshot.appendSections([Section.main])
87                  snapshot.appendItems(self.speakers, toSection: Section.main)
88                  self.dataSource?.apply(snapshot)
89              }
90          } else {
91              collectionView.reloadData()
92          }
93      }
```

Apply from background threads

```
84  DispatchQueue(label: "bq1", qos: .background).asyncAfter(deadline: .now() + 1, execute: {
85      var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
86      snapshot.appendSections([Section.main])
87      snapshot.appendItems(self.speakers, toSection: Section.main)
88      self.dataSource?.apply(snapshot, animatingDifferences: true, completion: {
89          debugPrint("first completed")
90      })
91  })
92
93  DispatchQueue(label: "bq2", qos: .background).asyncAfter(deadline: .now() + 1, execute: {
94      var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
95      snapshot.appendSections([Section.main])
96      snapshot.appendItems(self.speakers, toSection: Section.main)
97      self.dataSource?.apply(snapshot, animatingDifferences: true, completion: {
98          debugPrint("second completed")
99      })
100 })
101
102 DispatchQueue(label: "bq3", qos: .background).asyncAfter(deadline: .now() + 1, execute: {
103     var snapshot = NSDiffableDataSourceSnapshot<Section, Speaker>()
104     snapshot.appendSections([Section.main])
105     snapshot.appendItems(self.speakers, toSection: Section.main)
106     self.dataSource?.apply(snapshot, animatingDifferences: true, completion: {
107         debugPrint("third completed")
108     })
109 })
```

Apply from background threads

```
"first completed"
"third completed"
"second completed"
```

https://www.pinclipart.com/pindetail/ixJoRb_image-library-download-pinterest-disney-wiki-childhood-inside/

What about Supplementary views?

```
332  @available(iOS 13.0, tvOS 13.0, *)
333  open class UICollectionViewDiffableDataSource<SectionIdentifierType,
         ItemIdentifierType> : NSObject, UICollectionViewDataSource where
         SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {
```

What about Supplementary views?

```
332 @available(iOS 13.0, tvOS 13.0, *)
333 open class UICollectionViewDiffableDataSource<SectionIdentifierType,
        ItemIdentifierType> : NSObject, UICollectionViewDataSource where
        SectionIdentifierType : Hashable, ItemIdentifierType : Hashable {


337     public typealias SupplementaryViewProvider = (UICollectionView, String, IndexPath) ->
            UICollectionReusableView?
338
339     public var supplementaryViewProvider: UICollectionViewDiffableDataSource<SectionIdentifierType,
            ItemIdentifierType>.SupplementaryViewProvider?
```

What about Supplementary views?

```swift
import UIKit

final class SpeakerCollectionReusableView: UICollectionReusableView {

    @IBOutlet weak private var titleLabel: UILabel!

    override func prepareForReuse() {
        super.prepareForReuse()

        titleLabel.text = nil
    }
}

// MARK: - Display
extension SpeakerCollectionReusableView {

    func display(viewModel: String) {
        titleLabel.text = viewModel
    }
}
```

What about Supplementary views?

```
49      func setupCollectionView() {
50          collectionView.register(UINib(nibName: String(describing:
                SpeakerCollectionViewCell.self), bundle: nil),
                forCellWithReuseIdentifier: String(describing:
                SpeakerCollectionViewCell.self))
51
52          collectionView.register(UINib(nibName: String(describing:
                SpeakerCollectionReusableView.self), bundle: nil),
                forSupplementaryViewOfKind:
                UICollectionView.elementKindSectionHeader, withReuseIdentifier:
                String(describing: SpeakerCollectionReusableView.self))
53
54          collectionView.delegate = self
55          if #available(iOS 13.0, *) {
56              setupCollectionViewDataSource()
57          } else {
58              collectionView.dataSource = self
59          }
60      }
```

What about Supplementary views?

```
74          dataSource?.supplementaryViewProvider = { (collectionView, kind, indexPath) ->
                UICollectionReusableView? in
75              guard let headerView = collectionView.dequeueReusableSupplementaryView(ofKind:
                    kind, withReuseIdentifier: String(describing:
                    SpeakerCollectionReusableView.self), for: indexPath) as?
                    SpeakerCollectionReusableView else {
76                  fatalError("Could not dequeue SpeakerCollectionReusableView")
77              }
78
79              headerView.display(viewModel: "Header")
80
81              return headerView
82          }
```

Recap

- New API for our cell
- “New” API for our Supplementary Views
- Single source of truth (Snapshot)
- No custom performBatchUpdates
- Free updates

Try it out

- Apply for free animation
- iOS 13 only
- Compositional flow layout

GRAZIE

Jeroen Bakker - @JeroenBiOS

https://jeroenscode.com

https://github.com/Jeroenbb94/PragmaConference