Báo cáo tìm hiểu về AsyncDisplayKit – ComponentKit – IGListKit

Trần Đình Tôn Hiếu – hieutdt

iOS Fresher

* Tổng quan:

AsyncDisplayKit, ComponentKit và IGListKit đều là những UI framework hỗ trợ cho việc làm UI cho ứng dụng iOS trở nên tiện lợi hơn và trở nên mượt mà hơn, đạt hiệu quả cao hơn ở một số khía cạnh nhất định.

1. AsyncDisplayKit:

1.1. Giới thiệu về AsyncDisplayKit:

AsyncDisplayKit được tạo ra bởi team Paper của Facebook. Nó đến như là một câu trả lời cho câu hỏi cốt lõi mà Paper team phải đối mặt: “Làm sao để giữ cho main thread *clear* nhất có thể?”

Ngày nay, có rất nhiều ứng dụng có trải nghiệm người dùng phụ thuộc rất nhiều vào các thao tác liên tục và animations. Ít nhất, UI của ứng dụng cũng sẽ phải có một vài scroll view.

Các loại UI này phụ thuộc hoàn toàn vào main thread và rất nhạy cảm với việc main thread bị ngưng trệ khi phải xử lý một tác vụ nào đó tốn nhiều thời gian. Main thread bị tắc thì tức là frame bị sẽ bị đứng. Điều này sẽ làm cho trải nghiệm của người dùng tệ đi.

Một số tác vụ nặng mà main thread phải xử lý như: Đo đạc và layout, decode hình ảnh, drawing, tạo, huỷ đối tượng (ví dụ như UIView).

Nếu sử dụng AsyncDisplayKit đúng cách, AsyncDisplayKit cho phép bạn thực hiện tất cả các phép đo, bố cục và hiển thị mặc định là bất đồng bộ. Như vậy, ứng dụng của bạn sẽ được giảm một lượng rất đáng kể các tác vụ được thực hiện trên main thread.

Ngoài khả năng cải thiện về mặt hiệu suất, AsyncDisplayKit còn cung cấp các công cụ cho phép tạo ra các giao diện phức tạp một cách đơn giản hơn, số lượng code ít hơn.

1.2. ASDisplayNode:

ASDisplayNode là component cơ bản nhất của ASDisplayKit (ASDK), tương tự như UIView của UIKit.

Giống như việc UIView wrap CALayer thì ASDisplayNode về cơ bản là wrap một UIView.

Tuy nhiên, thay vì tạo và điều chỉnh một UIView phải thực hiện ở trên main thread, việc khởi tạo và điều chỉnh một ASDisplayNode có thể được thực hiện ở background và hiển thị mặc định là đồng thời (concurrently).

Khi sử dụng cùng với một trong số những node container, thuộc tính của một ASdisplayNode sẽ được set ở background thread, và lớp view/layer mà node đó wrap sẽ được tạo và được ASDisplayNode cache lại những thuộc tính.
Các API của ASDK được thiết kế rất tương đồng với UIKit nên việc sử dụng cũng sẽ trở nên khá đơn giản và gần gũi. Các component khác của ASDK sẽ kế thừa từ ASDisplayNode tương tự như với UIKit. Ví dụ như ta có ASTextNode, ASButtonNode, ASImageNode…

1.3. ASViewController:

Nếu như ASDisplayNode thay thế cho UIView, thì ASViewController được sử dụng tương ứng với một UIViewController. ASViewController có một số thay đổi so với UIViewController để phù hợp hơn với ASDisplayNode.
Một số ưu điểm của ASViewController:
- Tiết kiệm bộ nhớ: Khi một ASViewController được kết thúc, nó sẽ tự động tìm và thu hồi fetch data và display range của bất cứ node con nào của nó.
- Tính năng ASVisibility:….

1.4. Layout:

Layout API được tạo như một thay thế hiệu quả cho Auto Layout của UIKit – thứ trở nên xử lý quá nặng nề đối với các view hierarchies phức tạp. Layout API của ASDK có một số ưu điểm hơn so với Auto layout như sau:

- Nhanh: Nhanh hơn việc code layout thủ công và nhanh hơn rất nhiều so với auto layout.

- Asynchronous & Concurrent: Layout được xử lý ở background thread nên tương tác của user sẽ không bị gián đoạn.

- Rõ ràng: Layout được định nghĩa với kiểu cấu trúc dữ liệu bất biến. Nó làm cho layout code trở nên đơn giản hơn để phát triển, viết tài liệu, đánh giá, kiểm thử, sửa lỗi, mở rộng.

- Cacheable: Kết quả layout là cấu trúc dữ liệu bất biến để chúng có thể được tính toán trước ở background và được lưu trong bộ nhớ cache.

- Khả năng mở rộng: Dễ dàng chia sẻ mã nguồn giữa các lớp.

Layout API được lấy cảm hứng từ CSS Flexbox, nên sẽ có nhiều điểm tương đồng giữa cả hai.
Layout API bao gồm 2 khái niệm cơ bản là: LayoutSpecs và LayoutElement.
- LayoutSpecs: LayoutSpecs được xem như là container chứa các LayoutElement, quy định về vị trí và mối tương quan giữa các element với nhau. Layout API có cung cấp một số lớp con của LayoutSpecs hỗ trợ cho việc tạo các layout phức tạp.

- LayoutElement: LayoutSpecs sẽ chứa các LayoutElements. Cả ASDisplayNode và ASLayoutSpec đều conform protocol <ASLayoutElement>. Điều này có nghĩa là bạn có thể truyền cả ASDisplayNode hoặc ASLayoutSpec vào trong một ASLayoutSpec khác.

1.5. Node's lifecycle:

1.5.1 Node được quản lý bởi node container:

Node container chịu trách nhiệm về lifecycle của những node con mà nó quản lý. Các node container sẽ khởi tạo những node con của chúng khi cần và giải phóng khi chúng không còn được sử dụng. Texture cho rằng những node container sẽ chịu trách nhiệm quản lý hoàn toàn các node con của họ, và người dùng sẽ không lưu trữ lại những node con này và tác động đến lifecycle của chúng.

ASCollectionNode và AStableNode khởi tạo ASCellNode ngay khi chúng được thêm vào node container, thông qua reloadData hoặc những tao tác chèn. Tương tự như UICollectionView và UITableView, khởi tạo dữ liệu lần đầu tiên căn bản là reloadData mà không cần tập dữ liệu.

Tuy nhiên, nếu UICollectionView và UITableView thì những cells của chúng sẽ được reuse và reconfigured trước khi chúng hiển thị lên màn hình, ASCollectionNode và ASTableNode không tái sử dụng ASCellNodes. Điều này có nghĩa là ASCollectionNode và ASTableNode sẽ khởi tạo và tính toán layouts tất cả những ASCellNode cần thiết cùng lúc và thực hiện ở background thread.

Cũng chính vì vậy, ASCollectionNode và ASTableNode có thể cần phải mất một thời gian để tiến hành cập nhật. Vì vậy ta nên sử dụng "node block" API để khởi tạo các cell node ở background thread.

*** Deallocation:**

Như đã đề cập ở trên, bởi vì ASCellNode không được reuse, nên chúng có vòng đời dài hơn so với UICollectionViewCell và UITableViewCell. ASCellNode sẽ được thu hồi bộ nhớ khi chúng không còn được sử dụng hoặc bị xoá khỏi container node. Điều này sẽ dẫn đến việc phải thu hồi một lượng lớn hoặc rất lớn các cell node cùng một lúc → Có thể dẫn đến delay.

Để giải quyết vấn đề này, ASCollectionNode và ASTableNode giải phóng các biến đối tượng của chúng, tiêu biểu nhất là ASDataController, ở background thread với sự hỗ trợ của ASDeallocQueue. ASDataController là chủ nhân thực sự của tất cả các cell nodes - chứa tham chiếu strong đến tất cả cell nodes – được giải phóng thì tất cả các cell nodes cũng sẽ được giải phóng và không ảnh hưởng đến main thread.

1.5.2. Những node không được quản lý bởi container:

Đây là những node được tạo trực tiếp từ client code. Khi một node được thêm vào node cha, node cha sẽ giữ lại node đó cho đến khi nó được xoá khỏi node cha, hoặc node cha được

thu hồi. Tức là nếu như một node không được thu hồi bởi client code hoặc cũng không bị xoá khỏi node cha, thì vòng đời của nó sẽ gắn liền với vòng đời của node cha.

1.6. Một số ưu/khuyết điểm:

* Ưu điểm: Như đã trình bày ở trên, ưu điểm của ASDK có rất nhiều như:

- Hiệu suất cao, đảm bảo FPS cao khi thao tác trên UI.
- Không làm block main thread để đảm bảo trải nghiệm của người dùng.
- Việc layout trở nên đơn giản hơn bởi Layout API.
- Tương đồng với UIKit về mặt thiết kế API nên dễ tiếp cận và sử dụng.
- Document được viết rõ ràng, cụ thể và nhiều examples. Cộng đồng sử dụng lớn nên có thể tìm thấy nhiều nguồn tham khảo hơn.

* Khuyết điểm:

- Bộ nhớ.

2. ComponentKit:

2.1. Giới thiệu về ComponentKit:

Cũng tương tự như AsyncKit, ComponentKit là một UI Framework hỗ trợ cho việc hiển thị dữ liệu kiểu danh sách với nội dung phức tạp. Nó được tạo ra để hỗ trợ cho màn New Feeds của Facebook trên iOS, và được lấy cảm hứng từ React.

Một số ưu điểm của ComponentKit có thể kể đến như:

- Đơn giản và rõ ràng.
- Hiệu suất scroll cao: Có thể dễ dàng đạt 60FPS ngay cả đối với những layout phức tạp như màn New Feeds của Facebook. Các quá trình layout được thực hiện ở trên background (tương tự như AsyncKit).
- Tái sử dụng view: ComponentKit yêu cầu tất cả config cho một component đều phải được cài đặt lúc khai báo, vì vậy nên ComponentKit có thể tái sử dụng component đó mà không gây ra lỗi.
- Khả năng kết hợp: Bằng cách tạo ra nhiều component và tái sử dụng sẽ tạo ra được những UI phức tạp như New Feeds của Facebook mà không có component nào vượt quá 300 dòng code.

Một số cân nhắc khi sử dụng ComponentKit:

- Các giao diện không phải danh sách hoặc bảng thì không phù hợp với ComponentKit. ComponentKit được phát triển để làm việc với UICollectionView.
- ComponentKit được phát triển trên Objective-C++. Vậy nên không thể sử dụng cùng với Swift, vì Swift không thể tạo bridge với C++.

2.2. Component API:

Đối với ComponentKit thì CKComponent là lớp cơ sở. Các lớp con như CKButtonComponent, CKImageComponent đều là lớp con của CKComponent.

CKComponent cũng tương tự như ASDisplayNode, nó sẽ wrap một UIView và được khai báo cùng với UIView. Hàm tạo của nó như sau:

```
@interface CKComponent : NSObject
+ (instancetype)newWithView:(const CKComponentViewConfiguration &)view
                       size:(const CKComponentSize &)size;
@end
```

Một số lưu ý về CKComponent:

- CKComponent là hoàn toàn bất biến. Sẽ không có phương thức addSubComponent ở trong CKComponent.
- CKComponent có để được tạo ở bất kì thread nào.
- CKComponent sử dụng newWith… để khởi tạo thay vì alloc/init….
- Không nên tạo lớp con kế thừa từ CKComponent. Vì trong Objective-C không có "final" nên bất cứ phương thức nào cũng đều có thể bị override bởi lớp con. Hơn nữa, việc kế thừa như vậy khiến cho việc thay đổi lớp cha ít nhiều sẽ ảnh hưởng đến lớp con. Thay vào đó chúng ta nên kế thừa từ CKCompositeComponent và truyền Component cha vào hàm tạo.

2.3. CollectionView và Changeset API:

Một số ưu điểm khi sử dụng UICollectionView với ComponentKit có thể kể đến như:

- Tự động reuse: Ta không cần phải quan tâm đến việc reuse khi sử dụng UICollectionView vì ComponentKit đã hỗ trợ reuse.
- Hiệu suất scroll: Bởi vì các CKComponent có thể khởi tạo ở background, nên ComponentKit có thể sử dụng một cách tối đa việc chuyển các thao tác vào Background thread giúp đạt hiệu suất scroll tốt hơn.

Về việc cài đặt, ta sẽ sử dụng CKComponentProvider và CKCollectionViewDataSource để cài đặt một collectionView.

So với UICollectionViewDataSource, CKCollectionViewDataSource có một số ưu điểm như:

- Có thể sử dụng cho cả UITableView và UICollectionView.
- Tạo và layout ở trong background.

CKComponentProvider chịu trách nhiệm chuyển đổi model thành Component. Quá trình chuyển đổi này được định nghĩa là một phương thức của lớp conform CKComponentProvider. Lớp này sẽ được truyền cho CKCollectionViewDataSource như là một "nhà cung cấp component" và sẽ được DataSource gọi mỗi khi cần tạo một component từ model. Trong

trường hợp này, ta cần định nghĩa một Cell Component và trả về nó trong phương thức đã nói đến ở trên.

ComponentKit sử dụng Changeset API để cập nhật CollectionView. Những thay đổi trên models sẽ được chuyển đến cho dataSource, dataSource sẽ tính toán và áp dụng những thay đổi cho collection view. Điều này cũng tức là ta không thể sử dụng DataSource một cách trực tiếp – kế thừa CKComponentDataSource và implement các phương thức cập nhật dữ liệu. Trong tài liệu của ComponentKit có nói về việc sử dụng CKComponentDataSource trực tiếp, nhưng trong API của ComponentKit thì không cung cấp protocol CKComponentDataSource nữa nên ta không thể sử dụng cách này (có thể tài liệu chưa cập nhật).

2.4. Tạo Component:

Việc khởi tạo và layout một component khá là tương đồng với AsyncDisplayKit. Ta cũng sử dụng kết hợp các component với nhau để tạo ra component cuối cùng với UI theo ý muốn của bản thân.

2.5. Component Controller:

Component là những đối tượng bất biến, điều này cũng tức là, nếu như có một sự thay đổi và cần phải cập nhật thì một component mới sẽ được tạo ra thay thế cho component cũ bị xoá đi. Vậy nên, component là đối tượng có vòng đời ngắn, và vòng đời của nó không được kiểm soát.

Tuy nhiên, sẽ có lúc ta cần tạo ra một đối tượng với vòng đời dài, thì lúc này, ta sẽ sử dụng Component Controller để làm việc đó.

- Component không thể delegate bởi vì chúng có vòng đời ngắn, nhưng component controller thì có thể delegate.
- Với một task tốn nhiều thời gian, thì component không thể handle callback bởi vì nó có thể đã bị khởi tạo lại trong khoảng thời gian đó, nhưng Controller thì có thể handle callback.
- Bạn cần một đối tượng chứa một đối tượng khác thì đối tượng của bạn phải có vòng đời dài.

2.6. Đánh giá ưu/khuyết điểm:

* Ưu điểm: Từ những ưu điểm đã trình bày ở trên, ta có thể tổng hợp lại như sau:
    - Hiệu suất cao, FPS vẫn đạt 60FPS mặc dù layout phức tạp.
    - Khả năng tái sử dụng cao.
    - Mã nguồn rõ ràng, đơn giản bằng việc tách ra nhiều component.
    - Sẽ rất thân thiện với những ai đã tiếp xúc với React/React Native.

* Nhược điểm:

- Sử dụng Objective-C++ nên không thể dùng cho Swift.
- API được thiết kế không có nhiều điểm tương đồng với UIKit.
- Documents trình bày còn khó hiểu, chưa cập nhật kịp với thay đổi. Ít mã nguồn ví dụ.
- Ít tài liệu tham khảo.

3. IGListKit:

3.1. Giới thiệu IGListKit:

Điểm khác biệt rõ ràng nhất của IGListKit so với 2 framework ở trên chính là tính "flexible" của list. Ngay từ câu giới thiệu đầu tiên của IGListKit, ta cũng có thể thấy được những người viết documents đã khẳng định điều này như thế nào:

"A data-driven `UICollectionView` framework for building fast and flexible lists."

IGListKit được phát triển bởi đội kĩ sử của Instagram. Một số tính năng chính của IGListKit có thể kể đến như:

- Không bao giờ cần gọi lại peformBatchUpdate() hoặc reloadData().
- Kiến trúc tốt hơn về việc tái sử dụng các cells và components.
- Tạo ra collectionView với nhiều kiểu dữ liệu.
- Sử dụng thuật toán phân tách (Decoupled diffing algorithm).
- Được viết hoàn toàn bằng Objective-C nên hỗ trợ Swift 100%.

3.2. Diffing:

IGListKit sử dụng một thuật toán được điều chỉnh từ một bài báo khoa học tên *"A technique for isolating differences between files"* của Paul Heckel. Thuật toán này sử dụng kĩ thuật là chuỗi con chung dài nhất để tìm kiếm độ lệch tối thiểu giữa 2 mảng dữ liệu trong thời gian tuyến tính O(n). Nó sẽ tìm thấy tất cả các sự khác biệt về chèn, xoá, cập nhật giữa 2 mảng dữ liệu. IGListKit sẽ sử dụng kỹ thuật này để tiến hành cập nhật trạng thái cho CollectionView với thời gian nhỏ nhất.

Để sử dụng, model object của ta cần phải conform protocol <IGListDiffable> và cài đặt 2 phương thức là `diffIdentifier()` và `isEqual(toDiffableObject:)`

3.3. Flexible List:

Sự linh hoạt của IGListKit đến từ cách chúng ta tạo một UICollectionView bằng cách sử dụng nó. Việc chúng ta cần làm là khởi tạo một UICollectionView và IGListAdapter nằm trong lớp của chúng ta. Ta conform protocol IGListAdapterDataSource và set dataSource và collectionView cho IGListAdapter.

Trong IGListAdapterDataSource có 2 phương thức quan trọng đó là: objectsForListAdapter: và listAdapter:sectionControllerForObject:. 2 phương thức này sẽ định nghĩa xem collectionView của chúng ta bao gồm những model objects nào, và tương ứng với mỗi objectd đấy, SectionController của nó sẽ là gì. Như vậy tức là, với mỗi model object, có thể có một SectionController tương ứng. Điều này cũng đồng nghĩa là UICollectionView của chúng ta sẽ hiển thị được nhiều dạng row khác nhau trên cùng một CollectionView. Đây cũng chính là tính linh hoạt của IGListKit.

Mỗi SectionController là một IGSectionController, sẽ cung cấp thông tin về kích thước của cell và UICollectionCell tương ứng với model.

3.4. Đánh giá ưu/khuyết điểm:

* Ưu điểm:

- Tạo ra được giao diện danh sách linh hoạt, gồm nhiều loại dòng khác nhau trên một danh sách.
- Các API rõ ràng, việc cài đặt cũng trở nên khá đơn giản.
- Sử dụng thuật toán Diffing để tối ưu tốc độ cập nhật danh sách.
- Có thể sử dụng kết hợp với AsyncDisplayKit hoặc ComponentKit để tận dụng performance và khả năng layout của 2 framework kia.
- Documents viết tốt, nhiều examples. Cộng đồng sử dụng lớn với nhiều nguồn tham khảo.

* Nhược điểm: