Class

UICollectionViewLayout

An abstract base class for generating layout information for a collection view.

Declaration

```
class UICollectionViewLayout : NSObject
```

Overview

The job of a layout object is to determine the placement of cells, supplementary views, and decoration views inside the collection view's bounds and to report that information to the collection view when asked. The collection view then applies the provided layout information to the corresponding views so that they can be presented onscreen.

You must subclass `UICollectionViewLayout` in order to use it. Before you consider subclassing, though, you should look at the `UICollectionViewFlowLayout` class to see if it can be adapted to your layout needs.

Subclassing Notes

The main job of a layout object is to provide information about the position and visual state of items in the collection view. The layout object does not create the views for which it provides the layout. Those views are created by the collection view's data source. Instead, the layout object defines the position and size of visual elements based on the design of the layout.

Collection views have three types of visual elements that need to be laid out:

- **Cells** are the main elements positioned by the layout. Each cell represents a single data item in the collection. A collection view can have a single group of cells or it can divide those cells into multiple sections. The layout object's main job is to arrange the cells in the collection view's content area.
- **Supplementary views** present data but are different than cells. Unlike cells, supplementary views cannot be selected by the user. Instead, you use supplementary views to implement things like header and footer views for a given section or for the entire collection view. Supplementary views are optional and their use and placement is defined by the layout object.
- **Decoration views** are visual adornments that cannot be selected and are not inherently tied to the data of the collection view. Decoration views are another type of supplementary view. Like supplementary views, they are optional and their use and placement is defined by the layout object.

The collection view asks its layout object to provide layout information for these elements at many different times. Every cell and view that appears on screen is positioned using information from the layout object. Similarly, every time items are inserted into or deleted from the collection view, additional layout occurs for the items being added or removed. However, the collection view always

limits layout to the objects that are visible onscreen.

Methods to Override

Every layout object should implement the following methods:

- `collectionViewContentSize`
- `layoutAttributesForElements(in:)`
- `layoutAttributesForItem(at:)`
- `layoutAttributesForSupplementaryView(ofKind:at:)` (if your layout supports supplementary views)
- `layoutAttributesForDecorationView(ofKind:at:)` (if your layout supports decoration views)
- `shouldInvalidateLayout(forBoundsChange:)`

These methods provide the fundamental layout information that the collection view needs to place contents on the screen. Of course, if your layout does not support supplementary or decoration views, do not implement the corresponding methods.

When the data in the collection view changes and items are to be inserted or deleted, the collection view asks its layout object to update the layout information. Specifically, any item that is moved, added, or deleted must have its layout information updated to reflect its new location. For moved items, the collection view uses the standard methods to retrieve the item's updated layout attributes. For items being inserted or deleted, the collection view calls some different methods, which you should override to provide the appropriate layout information:

- `initialLayoutAttributesForAppearingItem(at:)`
- `initialLayoutAttributesForAppearingSupplementaryElement(ofKind:at:)`
- `initialLayoutAttributesForAppearingDecorationElement(ofKind:at:)`
- `finalLayoutAttributesForDisappearingItem(at:)`
- `finalLayoutAttributesForDisappearingSupplementaryElement(ofKind:at:)`
- `finalLayoutAttributesForDisappearingDecorationElement(ofKind:at:)`

In addition to these methods, you can also override the `prepare(forCollectionViewUpdates:)` to handle any layout-related preparation. You can also override the `finalizeCollectionViewUpdates()` method and use it to add animations to the overall animation block or to implement any final layout-related tasks.

Optimizing Layout Performance Using Invalidation Contexts

When designing your custom layouts, you can improve performance by invalidating only those parts of your layout that actually changed. When you change items, calling the `invalidateLayout()` method forces the collection view to recompute all of its layout information and reapply it. A better solution is to recompute only the layout information that changed, which is exactly what invalidation contexts allow you to do. An invalidation context lets you specify which parts of the layout changed. The layout object can then use that information to minimize the amount of data it recomputes.

To define a custom invalidation context for your layout, subclass the `UICollectionViewLayoutInvalidationContext` class. In your subclass, define custom properties that represent the parts of your layout data that can be recomputed independently. When you need to invalidate your layout at runtime, create an instance of your invalidation context subclass, configure the custom properties based on what layout information changed, and pass that object to your layout's `invalidateLayout(with:)` method. Your custom implementation of that method can use the information in

the invalidation context to recompute only the portions of your layout that changed.

If you define a custom invalidation context class for your layout object, you should also override the `invalidationContextClass` method and return your custom class. The collection view always creates an instance of the class you specify when it needs an invalidation context. Returning your custom subclass from this method ensures that your layout object always has the invalidation context it expects.

Topics

Initializing the Collection View

`init()`
Initializes the collection view layout object.

`init?(coder: NSCoder)`

Getting the Collection View Information

`var collectionView: UICollectionView?`
The collection view object currently using this layout object.

`var collectionViewContentSize: CGSize`
Returns the width and height of the collection view's contents.

Providing Layout Attributes

`class var layoutAttributesClass: AnyClass`

Returns the class to use when creating layout attributes objects.

`func prepare()`

Tells the layout object to update the current layout.

```
func layoutAttributesForElements(in: CGRect) -> [UICollectionViewLayoutAttributes]?
```

Returns the layout attributes for all of the cells and views in the specified rectangle.

```
func layoutAttributesForItem(at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the layout attributes for the item at the specified index path.

```
func layoutAttributesForInteractivelyMovingItem(at: IndexPath, withTargetPosition: CGPoint) -> UICollectionViewLayoutAttributes
```

Returns the layout attributes of an item when it is being moved interactively by the user.

```
func layoutAttributesForSupplementaryView(ofKind: String, at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the layout attributes for the specified supplementary view.

```
func layoutAttributesForDecorationView(ofKind: String, at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the layout attributes for the specified decoration view.

```
func targetContentOffset(forProposedContentOffset: CGPoint) -> CGPoint
```

Returns the content offset to use after an animated layout update or change.

```
func targetContentOffset(forProposedContentOffset: CGPoint, withScrollingVelocity: CGPoint) -> CGPoint
```

Returns the point at which to stop scrolling.

Responding to Collection View Updates

```
func prepare(forCollectionViewUpdates: [UICollectionViewUpdateItem])
```

Notifies the layout object that the contents of the collection view are about to change.

```
func finalizeCollectionViewUpdates()
```

Performs any additional animations or clean up needed during a collection view update.

```
func indexPathsToInsertForSupplementaryView(ofKind: String) -> [IndexPath]
```

Returns an array of index paths for the supplementary views you want to add to the layout.

```
func indexPathsToInsertForDecorationView(ofKind: String) -> [IndexPath]
```

Returns an array of index paths representing the decoration views to add.

```
func initialLayoutAttributesForAppearingItem(at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the starting layout information for an item being inserted into the collection view.

```
func initialLayoutAttributesForAppearingSupplementaryElement(ofKind: String, at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the starting layout information for a supplementary view being inserted into the collection view.

```
func initialLayoutAttributesForAppearingDecorationElement(ofKind: String, at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the starting layout information for a decoration view being inserted into the collection view.

```
func indexPathsToDeleteForSupplementaryView(ofKind: String) -> [IndexPath]
```

Returns an array of index paths representing the supplementary views to remove.

```
func indexPathsToDeleteForDecorationView(ofKind: String) -> [IndexPath]
```

Returns an array of index paths representing the decoration views to remove.

```
func finalLayoutAttributesForDisappearingItem(at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the final layout information for an item that is about to be removed from the collection view.

```
func finalLayoutAttributesForDisappearingSupplementaryElement(ofKind: String, at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the final layout information for a supplementary view that is about to be removed from the collection view.

```
func finalLayoutAttributesForDisappearingDecorationElement(ofKind: String, at: IndexPath) -> UICollectionViewLayoutAttributes?
```

Returns the final layout information for a decoration view that is about to be removed from the collection view.

```
func targetIndexPath(forInteractivelyMovingItem: IndexPath, withPosition: CGPoint) -> IndexPath
```

Returns the index path to for an item when it is at the specified location in the collection view's bounds.

Invalidating the Layout

```
func invalidateLayout()
```
Invalidates the current layout and triggers a layout update.

```
func invalidateLayout(with: UICollectionViewLayoutInvalidation
Context)
```
Invalidates the current layout using the information in the provided context object.

```
class var invalidationContextClass: AnyClass
```
Returns the class to use when creating an invalidation context for the layout.

```
func shouldInvalidateLayout(forBoundsChange: CGRect) -> Bool
```
Asks the layout object if the new bounds require a layout update.

```
func invalidationContext(forBoundsChange: CGRect) -> UICollection
ViewLayoutInvalidationContext
```
Returns a context object that defines the portions of the layout that should change when a bounds change occurs.

```
func shouldInvalidateLayout(forPreferredLayoutAttributes:
UICollectionViewLayoutAttributes, withOriginalAttributes:
UICollectionViewLayoutAttributes) -> Bool
```
Asks the layout object if changes to a self-sizing cell require a layout update.

```
func invalidationContext(forPreferredLayoutAttributes: UICollection
ViewLayoutAttributes, withOriginalAttributes: UICollectionView
LayoutAttributes) -> UICollectionViewLayoutInvalidationContext
```
Returns a context object that identifies the portions of the layout that should change in response to dynamic cell changes.

```
func invalidationContext(forInteractivelyMovingItems: [IndexPath],
withTargetPosition: CGPoint, previousIndexPaths: [IndexPath],
previousPosition: CGPoint) -> UICollectionViewLayoutInvalidation
Context
```
Returns a context object that identifies the items that are being interactively moved in the layout.

```
func invalidationContextForEndingInteractiveMovementOfItems(toFinal
IndexPaths: [IndexPath], previousIndexPaths: [IndexPath], movement
Cancelled: Bool) -> UICollectionViewLayoutInvalidationContext
```
Returns a context object that identifies the items that were moved

Coordinating Animated Changes

`func prepare(forAnimatedBoundsChange: CGRect)`
Prepares the layout object for animated changes to the view's bounds or the insertion or deletion of items.

`func finalizeAnimatedBoundsChange()`
Cleans up after any animated changes to the view's bounds or after the insertion or deletion of items.

Transitioning Between Layouts

`func prepareForTransition(from: UICollectionViewLayout)`
Tells the layout object to prepare to be installed as the layout for the collection view.

`func prepareForTransition(to: UICollectionViewLayout)`
Tells the layout object that it is about to be removed as the layout for the collection view.

`func finalizeLayoutTransition()`
Tells the layout object to perform any final steps before the transition animations occur.

Registering Decoration Views

`func register(AnyClass?, forDecorationViewOfKind: String)`
Registers a class for use in creating decoration views for a collection view.

`func register(UINib?, forDecorationViewOfKind: String)`
Registers a nib file for use in creating decoration views for a collection view.

Supporting Right-To-Left Layouts

`var developmentLayoutDirection: UIUserInterfaceLayoutDirection`
The direction of the language you used when designing your custom layout.

`var flipsHorizontallyInOppositeLayoutDirection: Bool`
A Boolean value indicating whether the horizontal coordinate system is automatically flipped at appropriate times.

Relationships

Inherits From

`NSObject`

Conforms To

CVarArg
Equatable
Hashable
NSCoding

See Also

Layouts

class UICollectionViewFlowLayout
A concrete layout object that organizes items into a grid with optional header and footer views for each section.

class UICollectionViewTransitionLayout
A special type of layout object that lets you implement behaviors when changing from one layout to another in your collection view.

class UICollectionViewLayoutAttributes
A layout object that manages the layout-related attributes for a given item in a collection view.

{}Customizing Collection View Layouts
Customize a view layout by changing the size of cells in the flow or implementing a mosaic style.