Application, ViewController

Outline

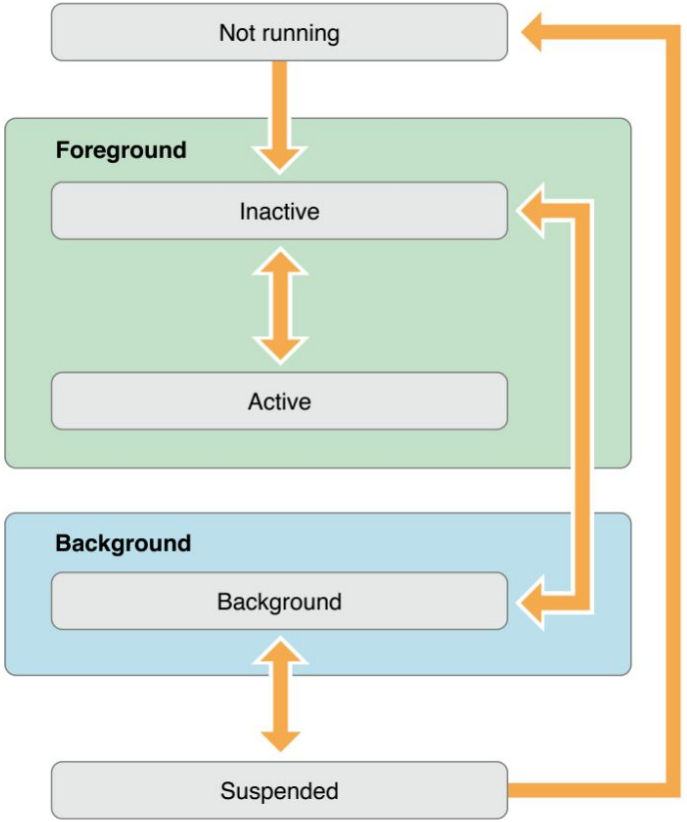
1. Application
2. ViewController

1. Application

1. Overview
2. Scene-Based Life-Cycle Events
3. App-Based Life-Cycle Events

1.1 Overview

- Application lifecycle:



1.1 Overview (cont)

- The current state of your app determines what it can and cannot do at any time. For example, a foreground app has the user's attention, so it has priority over system resources, including the CPU. By contrast, a background app must do as little work as possible, and preferably nothing, because it is offscreen. As your app changes from state to state, you must adjust its behavior accordingly.

- When your app's state changes, `UIKit` notifies you by calling methods of the appropriate delegate object:

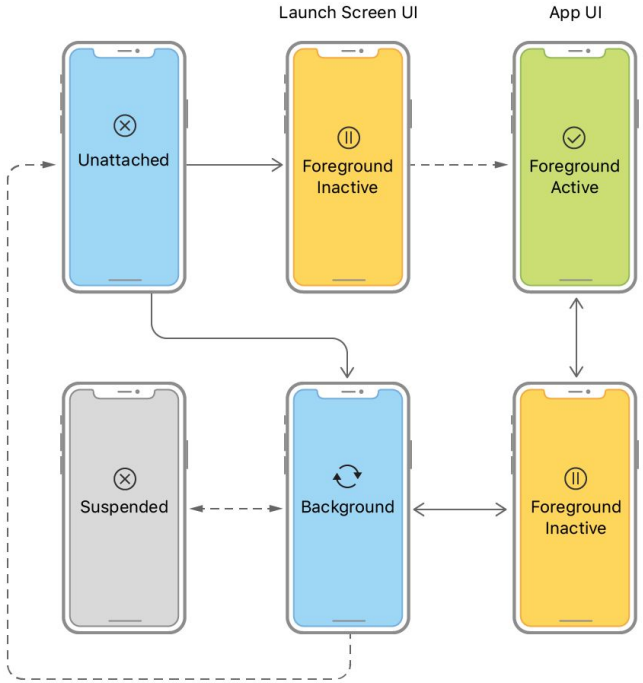
  - In iOS 13 and later, use `UISceneDelegate` objects to respond to life-cycle events in a scene-based app.
  - In iOS 12 and earlier, use the `UIApplicationDelegate` object to respond to life-cycle events.

1.2 Scene-Based Life-Cycle Events

- If your app supports scenes, UIKit delivers separate life-cycle events for each. A scene represents one instance of your app's UI running on a device. The user can create multiple scenes for each app, and show and hide them separately. Because each scene has its own life cycle, each can be in a different state of execution.

- Scene support is an opt-in feature. To enable basic support, add the `UIApplicationSceneManifest` key to your app's `Info.plist` file

1.2 Scene-Based Life-Cycle Events (cont)



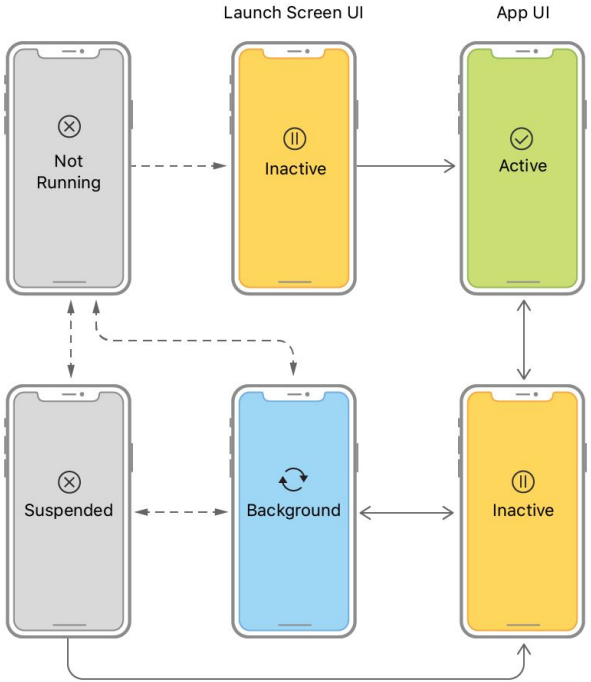
- Use scene transitions to perform the following tasks:
  - When UIKit connects a scene to your app, configure your scene's initial UI and load the data your scene needs.
  - When transitioning to the foreground-active state, configure your UI and prepare to interact with the user.
  - Upon leaving the foreground-active state, save data and quiet your app's behavior.
  - Upon entering the background state, finish crucial tasks, free up as much memory as possible, and prepare for your app snapshot.
  - At scene disconnection, clean up any shared resources associated with the scene.
  - In addition to scene-related events, you must also respond to the launch of your app using your UIApplicationDelegate object.

1.3 App-Based Life-Cycle Events

- In iOS 12 and earlier, and in apps that don't support scenes, UIKit delivers all life-cycle events to the UIApplicationDelegate object.

- The app delegate manages all of your app’s windows, including those displayed on separate screens. As a result, app state transitions affect your app's entire UI, including content on external displays.

1.3 App-Based Life-Cycle Events (cont)



- Use app transitions to perform the following tasks:
  - At launch, initialize your app's data structures and UI.
  - At activation, finish configuring your UI and prepare to interact with the user.
  - Upon deactivation, save data and quiet your app's behavior.
  - Upon entering the background state, finish crucial tasks, free up as much memory as possible, and prepare for your app snapshot.
  - At termination, stop all work immediately and release any shared resources.

2. ViewController

1. Overview
2. ViewController Life-Cycle
3. View Management
4. Memory Management
5. State Preservation and Restoration

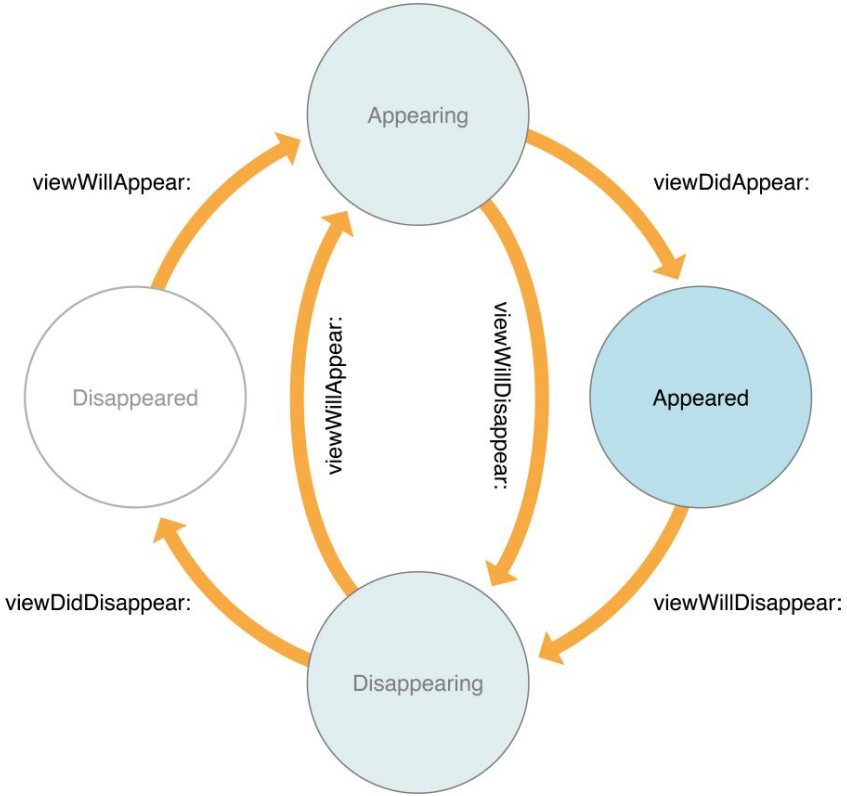
2.1 Overview

- The `UIViewController` class defines the shared behavior that is common to all view controllers. You rarely create instances of the `UIViewController` class directly. Instead, you subclass `UIViewController` and add the methods and properties needed to manage the view controller's view hierarchy.

- A view controller’s main responsibilities include the following:

  - Updating the contents of the views, usually in response to changes to the underlying data.
  - Responding to user interactions with views.
  - Resizing views and managing the layout of the overall interface.
  - Coordinating with other objects—including other view controllers—in your app.

2.2 ViewController Life-Cycle



2.3 View Management

- Each view controller manages a view hierarchy, the root view of which is stored in the view property of this class. The root view acts primarily as a container for the rest of the view hierarchy.

- The size and position of the root view is determined by the object that owns it, which is either a parent view controller or the app's window.

- The view controller that is owned by the window is the app's root view controller and its view is sized to fill the window.

2.3 View Management (cont)

- View controllers load their views lazily. Accessing the view property for the first time loads or creates the view controller's views. There are several ways to specify the views for a view controller:

  - ➔ Specify the view controller and its views in your app's Storyboard.
  - ➔ Specify the views for a view controller using a Nib file.
  - ➔ Specify the views for a view controller using the `loadView()` method.

2.4 Memory Management

- Memory is a critical resource in iOS, and view controllers provide built-in support for reducing their memory footprint at critical times.

- The UIViewController class provides some automatic handling of low-memory conditions through its `didReceiveMemoryWarning()` method, which releases unneeded memory.

2.5 State Preservation and Restoration

- If you assign a value to the view controller's `restorationIdentifier` property, the system may ask the view controller to encode itself when the app transitions to the background.

- When preserved, a view controller preserves the state of any views in its view hierarchy that also have restoration identifiers. View controllers do not automatically save any other state.

- If you are implementing a custom container view controller, you must encode any child view controllers yourself. Each child you encode must have a unique restoration identifier.

Question & Answer?