Architecting Your App for Multiple Windows

Janum Trivedi, UIKit
Changes to app lifecycle

Using the scene delegate

Architecture
Changes to app lifecycle
Using the scene delegate
Architecture
Changes to app lifecycle
Using the scene delegate
Architecture
App Delegate Responsibilities

iOS 12 and earlier
App Delegate Responsibilities
iOS 12 and earlier

App Delegate

Process Lifecycle

- App Launched
- App Terminated
- ...

App Delegate Responsibilities

iOS 12 and earlier

Diagram:
- **App Delegate**
  - **Process Lifecycle**
    - App Launched
    - App Terminated
  - **UI Lifecycle**
    - Entered Foreground
    - Became Active

...
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  var window: UIWindow?

  func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    // Do some global setup
    Database.connect()

    // Set up the UI...
    window = UIWindow()

  }
}
@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        // Do some global setup
        Database.connect()

        // Set up the UI...
        window = UIWindow()
    }
}
@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        // Do some global setup
        Database.connect()

        // Set up the UI...
        window = UIWindow()
    }
}
App Delegate Changes
iOS 13

App Delegate

Process Lifecycle
- App Launched
- App Terminated

UI Lifecycle
- Entered Foreground
- Became Active
App Delegate Changes
iOS 13

- App Delegate
  - Process Lifecycle
    - App Launched
    - App Terminated
  - UI Lifecycle
    - Entered Foreground
    - Became Active
App Delegate Changes
iOS 13

App Delegate
- Process Lifecycle
  - App Launched
  - App Terminated

Scene Delegate
- UI Lifecycle
  - Entered Foreground
  - Became Active

NEW
## iOS 13

<table>
<thead>
<tr>
<th>UIApplicationDelegate</th>
<th>UISceneDelegate</th>
</tr>
</thead>
<tbody>
<tr>
<td>application:willEnterForeground</td>
<td>scene:willEnterForeground</td>
</tr>
<tr>
<td>application:didEnterBackground</td>
<td>scene:didEnterBackground</td>
</tr>
<tr>
<td>application:willResignActive</td>
<td>scene:willResignActive</td>
</tr>
<tr>
<td>application:didBecomeActive</td>
<td>scene:didBecomeActive</td>
</tr>
</tbody>
</table>
**Session Lifecycle**

*iOS 13*

- **App Delegate**
  - Process Lifecycle
    - App Launched
    - App Terminated

- **Scene Delegate**
  - UI Lifecycle
    - Entered Foreground
    - Became Active
Session Lifecycle
iOS 13

App Delegate
- Process Lifecycle
  - Session Lifecycle
    - Session Created
    - Session Discarded
  - Scene Delegate
    - UI Lifecycle
      - Entered Foreground
      - Became Active
App Delegate

didFinishLaunching

configurationForSession
Configuring New Sessions

```swift
func application(_ application: UIApplication,
    configurationForConnecting connectingSceneSession: UISceneSession,
    options: UIScene.ConnectionOptions) -> UISceneConfiguration
```

Select a scene `configuration`

Provides user activities, URLs

Static and dynamic definition
func application(_ application: UIApplication,
    configurationForConnecting connectingSceneSession: UISceneSession,
    options: UIScene.ConnectionOptions) -> UISceneConfiguration

Select a scene *configuration*

Provides user activities, URLs

Static and dynamic definition
class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {

        // First check the options...
        return UISceneConfiguration(name: "Default", sessionRole: connectingSceneSession.role)
    }
}
// Configuring incoming scenes

class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) -> UISceneConfiguration {

        // First check the options...
        return UISceneConfiguration(name: "Default", sessionRole: connectingSceneSession.role)
    }
}
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options: .ConnectionOptions)
    {
        window = UIWindow(windowScene: scene as! UIWindowScene)

        if let activity = options.userActivities.first ?? session.stateRestorationActivity {
            configure(window: window, with: activity)
        }
    }
}
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?

    func scene(_ scene:UIScene, willConnectTo session:UISceneSession, options:.ConnectionOptions) {

        window = UIWindow(windowScene: scene as! UIWindowScene)

        if let activity = options.userActivities.first ?? session.stateRestorationActivity {
            configure(window:window, with:activity)
        }
    }
}
App Delegate

- didFinishLaunching
- configurationForSession

Scene Delegate

- willConnectToSession
Scene Delegate

willConnectToSession
Scene Delegate

- willConnectToSession
- willResignActive
- didEnterBackground
- didDisconnect
Scene Disconnection

```swift
func sceneDidDisconnect(_ scene: UIScene)
```

System is releasing the scene
May be called any time
Release associated resources
The scene may return!
Scene Disconnection

```swift
func sceneDidDisconnect(_ scene: UIScene)
```

System is releasing the scene
May be called any time
Release associated resources
The scene may return!
didEnterBackground

didDisconnect

App Delegate

didDiscardSceneSession
Cleaning up Discarded Sessions

```swift
func application(_ application: UIApplication, 
didDiscardSceneSessions sceneSessions: Set<UISceneSession>)
```

For permanently discarded sessions
Delete associated data
May be called after next launch
Cleaning up Discarded Sessions

```swift
func application(_ application: UIApplication,
    didDiscardSceneSessions sceneSessions: Set<UISceneSession>)
```

For permanently discarded sessions

Delete associated data

May be called after next launch
Changes to app lifecycle
Using the scene delegate
Architecture
State Restoration
Per-Scene State Restoration

func stateRestorationActivity(for scene: UIScene) -> NSUserActivity?

Called on scene background
Encode state via NSUserActivity
Data protection
Per-Scene State Restoration

```swift
func stateRestorationActivity(for scene: UIScene) -> NSUserActivity?
```

Called on scene background

Encode state via NSUserActivity

Data protection
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    func stateRestorationActivity(for scene: UIScene) -> NSUserActivity? {
        let currentActivity = fetchCurrentUserActivity(for: self.window)
        return currentActivity
    }

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options: .ConnectionOptions)
    if let restorationActivity = session.stateRestorationActivity {
        self.configure(window: window, with: restorationActivity)
    }
}
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    func stateRestorationActivity(for scene: UIScene) -> NSUserActivity? {
        let currentActivity = fetchCurrentUserActivity(for: self.window)
        return currentActivity
    }

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options: .ConnectionOptions)
    if let restorationActivity = session.stateRestorationActivity {
        self.configure(window: window, with: restorationActivity)
    }
}
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    func stateRestorationActivity(for scene: UIScene) -> NSUserActivity? {
        let currentActivity = fetchCurrentUserActivity(for: self.window)
        return currentActivity
    }

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options: .ConnectionOptions)
    {
        if let restorationActivity = session.stateRestorationActivity {
            self.configure(window: window, with: restorationActivity)
        }
    }
}
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    func stateRestorationActivity(for scene: UIScene) -> NSUserActivity? {
        let currentActivity = fetchCurrentUserActivity(for: self.window)
        return currentActivity
    }

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options: .ConnectionOptions)
    {
        if let restorationActivity = session.stateRestorationActivity {
            self.configure(window: window, with: restorationActivity)
        }
    }
}
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    func stateRestorationActivity(for scene: UIScene) -> NSUserActivity? {
        let currentActivity = fetchCurrentUserActivity(for: self.window)
        return currentActivity
    }

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options: .ConnectionOptions)
        if let restorationActivity = session.stateRestorationActivity {
        self.configure(window: window, with: restorationActivity)
    }
}
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    func stateRestorationActivity(for scene: UIScene) -> NSUserActivity? {
        let currentActivity = fetchCurrentUserActivity(for: self.window)
        return currentActivity
    }

    func scene(_ scene: UIScene, willConnectTo session:UISceneSession, options:.ConnectionOptions) {
        if let restorationActivity = session.stateRestorationActivity {
            self.configure(window: window, with: restorationActivity)
        }
    }
}
Keeping Scenes in Sync
Hey Giovanni!
Want to grab lunch?

Sure, let's head out in 10.
Hey Giovanni!
Want to grab lunch?

Sure, let's head out in 10.
Button tap

View Controller

Update views

Model Controller

Update model
Button tap → View Controller

View Controller → Model Controller

Model Controller → Update model

Update model → View Controller

Update views → View Controller

Old state
Button tap

View Controller

Model Controller

View Controller

Update model
class ChatViewController: UIViewController {

    @objc func didEnterMessage(sender: UITextField) {
        let message = Message(text: sender.text)

        // Update views
        self.animateNewRow(for: message)
        self.updateBadgeCount()

        // Update the model
        ChatModelController.shared.add(message: message)
    }
}
class ChatViewController: UIViewController {

    @objc func didEnterMessage(sender: UITextField) {

        let message = Message(text: sender.text)

        // Update the model
        ChatModelController.shared.add(message: message)
    }
}

class ChatModelController {

    static let shared = ChatModelController()

    func add(message: Message) {
        saveToDisk(message)
    }
}
How will we send the update down?
enum UpdateEvent {

}

enum UpdateEvent {
    case NewMessage(message: Message)
}
enum UpdateEvent {
    case NewMessage(message: Message)

    static let NewMessageNotificationName = Notification.Name(rawValue: "NewMessage")

    func post() {
        // Notify subscribers
    }
}
enum UpdateEvent {
    case NewMessage(message: Message)
}

static let NewMessageNotificationName = Notification.Name(rawValue: "NewMessage")

func post() {
    // Notify subscribers
    switch self {
    case .NewMessage(message: _): NotificationCenter.default.post(
        name: UpdateEvent.NewMessageNotificationName,
        object: self
    )
    }
}
class ChatModelController {

    static let shared = ChatModelController()

    func add(message: Message) {
        saveToDisk(message)

        let event = UpdateEvent.NewMessage(message: message)
        event.post()
    }
}

class ChatModelController {

    static let shared = ChatModelController()

    func add(message: Message) {
        saveToDisk(message)

        let event = UpdateEvent.NewMessage(message: message)
        event.post()
    }
}
class ChatViewController: UIViewController {

    override func viewDidLoad() {
        NotificationCenter.default.addObserver(selector: ..., name: .NewMessageNotificationName)
    }

}
class ChatViewController: UIViewController {

    override func viewDidLoad() {
        NotificationCenter.default.addObserver(selector: ..., name: .NewMessageNotificationName)
    }
}

}

class ChatViewController: UIViewController {

    override func viewDidLoad() {
        NotificationCenter.default.addObserver(selector: ..., name: .NewMessageNotificationName)
    }

    @objc func handle(notification: Notification) {
        let event = notification.object as! UpdateEvent

        switch event {
        case .NewMessage(message: newMessage):
            // Update the UI
            self.animateNewRow(for: newMessage)
            self.updateBadgeCount()
        }
    }
}

class ChatViewController: UIViewController {

override func viewDidLoad() {
    NotificationCenter.default.addObserver(selector: ..., name: .NewMessageNotificationName)
}

@objc func handle(notification: Notification) {
    let event = notification.object as! UpdateEvent

    switch event {
    case .NewMessage(message: newMessage):
        // Update the UI
        self.animateNewRow(for: newMessage)
        self.updateBadgeCount()
    }
}
}
class ChatViewController: UIViewController {

    override func viewDidLoad() {
        NotificationCenter.default.addObserver(selector: ..., name: .NewMessageNotificationName)
    }

    @objc func handle(notification: Notification) {
        let event = notification.object as! UpdateEvent
        switch event {
            case .NewMessage(message: newMessage):
                // Update the UI
                self.animateNewRow(for: newMessage)
                self.updateBadgeCount()
        }
    }
}
class ChatViewController: UIViewController {

    override func viewDidLoad() {
        NotificationCenter.default.addObserver(selector: ..., name: .NewMessageNotificationName)
    }

    @objc func handle(notification: Notification) {
        let event = notification.object as! UpdateEvent

        switch event {
        case .NewMessage(message: newMessage):
            // Update the UI
            self.animateNewRow(for: newMessage)
            self.updateBadgeCount()
        }
    }
}

Giovanni Tarducci

Hey Giovanni!
Want to grab lunch?

Michael, Jamie

Andrea Kopitz

Sure, let's head out in 10.

Andrea Kopitz

Sure, let's head out in 10.
Giovanni Tarducci

Michael, Jamie

Andrea Kopitz

Janum Trivedi at 6/5/19
Hey Giovanni! Want to grab lunch?

Andrea Kopitz

Sure, let's head out in 10.
Review

App vs. scene delegate

State restoration

Keeping scenes in sync
More Information

developer.apple.com/wwdc19/258
Apple WWDC19