The View Controller Hierarchy

The relationships among your app's view controllers define the behaviors required of each view controller. UIKit expects you to use view controllers in prescribed ways. Maintaining the proper view controller relationships ensures that automatic behaviors are delivered to the correct view controllers when they are needed. If you break the prescribed containment and presentation relationships, portions of your app will stop behaving as expected.

앱의 뷰 컨트롤러간의 관계는 각 뷰 컨트롤러에 필요한 기능을 정의합니다. UIKit은 여러분이 항상 지정된 방식으로 뷰 컨트롤러를 사용하기를 바랍니다. 뷰 컨트롤러간의 관계를 적절히 유지하는 것은 필요할 때 자동으로 제공되는 기능이 적절한 뷰 컨트롤러에 전달되는 데에 필수적입니다. 만약 이러한 관계를 잘못 형성한다면 앱의 일부가 예상대로 작동하지 않을 것입니다.

The Root View Controller

The root view controller is the anchor of the view controller hierarchy. Every window has exactly one root view controller whose content fills that window. The root view controller defines the initial content seen by the user. Figure 2-1 shows the relationship between the root view controller and the window. Because the window has no visible content of its own, the view controller's view provides all of the content.

루트 뷰 컨트롤러는 뷰 컨트롤러 계층의 중요한 역할을 수행합니다. 모든 윈도우에는 해당 윈도우를 가득 채우는 하나의 루트 뷰 컨트롤러를 가집니다. 루트 뷰 컨트롤러는 사용자가 보는 초기 컨텐츠를 정의하고 화면에 보여줄 수 있습니다. 그림 2-1은 루트 뷰 컨트롤러와 윈도우 간의 관계를 보여주고 있습니다. 윈도우는 자체적으로 화면에 보이는 컨텐츠가 없으므로 뷰 컨트롤러의 뷰가 모든 컨텐츠를 제공합니다.

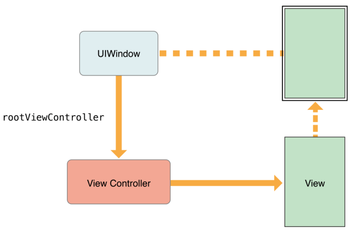

Figure 2-1 The root view controller

The root view controller is accessible from the `rootViewController` property of the `UIWindow` object. When you use storyboards to configure your view controllers, UIKit sets the value of that property automatically at launch time. For windows you create programmatically, you must set the root view controller yourself.

루트 뷰 컨트롤러는 `UIWindow` 객체의 `rootViewController` 프로퍼티로 접근할 수 있습니다. 뷰 컨트롤러를 스토리보드를 이용해 만들 때, UIKit은 앱 실행 시 루트 뷰 컨트롤러의 프로퍼티에 적절한 값을 자동으로 설정합니다. 윈도우를 코드로 생성하면 루트 뷰 컨트롤러를 직접 설정하고 생성하여야 합니다.

Container View Controllers

Container view controllers let you assemble sophisticated interfaces from more manageable and reusable pieces. A container view controller mixes the content of one or more child view controllers together with optional custom views to create its final interface. For example, a `UINavigationController` object displays the content from a child view controller together with a navigation bar and optional toolbar, which are managed by the navigation controller. UIKit includes several container view controllers, including `UINavigationController`, `UISplitViewController`, and `UIPageViewController`.

컨테이너 뷰 컨트롤러는 관리하기 쉽고 재사용가능한 정교한 인터페이스를 모아서 사용할 수 있도록 해줍니다. 컨테이너 뷰 컨트롤러는 하나 또는 그 이상의 자식 뷰 컨트롤러의 내용과 커스텀 뷰와 함께 사용하여 최종 인터페이스를 생성합니다. 예를 들어, `UINavigationController` 객체는 네비게이션 바와 옵셔널 툴바와 함께 자식 뷰 컨트롤러의 컨텐츠를 화면에 보여줍니다. UIKit은 `UINavigationController`, `UISplitViewController`, `UIPageViewController`를 포함한 여러 컨테이너 뷰 컨트롤러를 가지고 있습니다.

A container view controller's view always fills the space given to it. Container view controllers are often installed as root view controllers in a window (as shown in Figure 2-2), but they can also be presented modally or installed as children of other containers. The container is responsible for positioning its child views appropriately. In the figure, the container places the two child views side by side. Although it depends on the container interface, child view controllers may have minimal knowledge of the container and any sibling view controllers.

컨테이너 뷰 컨트롤러의 뷰는 항상 지정된 공간을 가득 채우게 됩니다. 컨테이너 뷰 컨트롤러는 대개 윈도우의 루트 뷰 컨트롤러로 (그림 2-2처럼) 동작하게 되지만, 모달방식으로 제공되거나 다른 컨테이너 뷰 컨트롤러의 자식으로 동작할 수도 있습니다. 컨테이너는 자기가 가지고 있는 자식 뷰들이 적절한 위치를 가질 수 있도록 해야 합니다. 아래의 그림에서 컨테이너는 자식 뷰를 나란히 배치하고 있습니다. 컨테이너 인터페이스 따라 다르지만, 자식 뷰 컨트롤러는 컨테이너, 모든 형제 뷰 컨트롤러에 대한 정보를 최소한만 가질 수 있도록 해야 합니다.

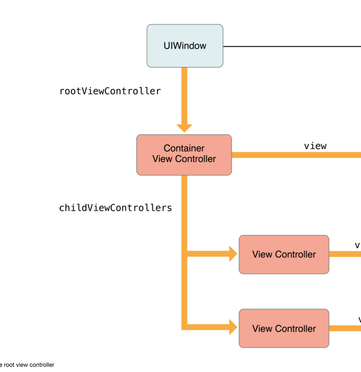

Figure 2-2 A container acting as the root view controller

Because a container view controller manages its children, UIKit defines rules for how you set up those children in custom containers. For detailed information about how to create a custom container view controller, see Implementing a Container View Controller.

컨테이너 뷰 컨트롤러는 자식을 관리하므로, UIKit은 커스텀 컨테이너에 어떻게 자식을 받아들이고 설정할지 그 방법에 대한 규칙을 정의하였습니다. 커스텀 컨테이너 뷰 컨트롤러를 생성하는 자세한 정보는 Implementing a Container View Controller 을 참고하세요.

Presented View Controllers

Presenting a view controller replaces the current view controller's contents with those of a new one, usually hiding the previous view controller's contents. Presentations are most often used for displaying new content modally. For example, you might present a view controller to gather input from the user. You can also use them as a general building block for your app's interface.

뷰 컨트롤러를 화면에 표시하는 것은 현재 보이고 있는 뷰 컨트롤러의 컨텐츠를 새 뷰 컨트롤러의 내용으로 바꾸는 것을 의미합니다. 대개 전 뷰컨트롤러의 내용을 숨기면서 덮이죠. 프리젠테이션은 자주 새로운 컨텐츠를 모달 방식으로 표시하는 것으로 사용됩니다. 예를 들어, 사용자에게 입력을 받기 위한 뷰 컨트롤러를 화면에 표시할 수 있습니다. 또한 앱의 인터페이스를 위한 일반적인 빌딩 block으로 사용될 수 있습니다.

When you present a view controller, UIKit creates a relationship between the *presenting view controller* and the *presented view controller*, as shown in Figure 2-3. (There is also a reverse relationship from the presented view controller back to its presenting view controller.) These relationships form part of the view controller hierarchy and are a way to locate other view controllers at runtime.

뷰 컨트롤러를 화면에 표시할 때, UIKit은 아래의 그림 2-3처럼 *presenting view controller*, *presented view controller*사이의 관계를 만듭니다. 이러한 관계는 뷰 컨트롤러의 계층구조의 일부분을 형성하며 앱 실행 시 다른 뷰 컨트롤러가 어디에 있는지 찾는 방법으로 사용될 수 있습니다.

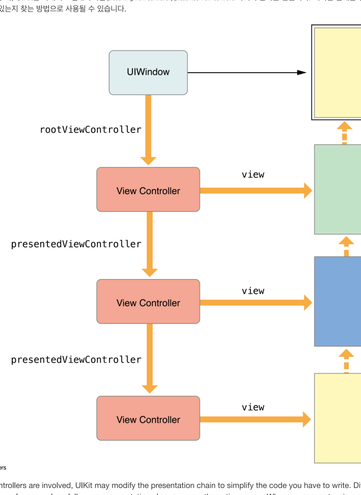

Figure 2-3 Presented view controllers

When container view controllers are involved, UIKit may modify the presentation chain to simplify the code you have to write. Different presentation styles have different rules for how they appear onscreen—for example, a full-screen presentation always covers the entire screen. When you present a view controller, UIKit looks for a view controller that provides a suitable context for the presentation. In many cases, UIKit chooses the nearest container view controller but it might also choose the window's root view controller. In some cases, you can also tell UIKit which view controller defines the presentation context and should handle the presentation.

컨테이너 뷰 컨트롤러가 사용될 때, UIKit은 프리젠테이션 체인을 수정하여 작성해야 하는 코드의 양을 줄일 수 있습니다. 프리젠테이션 스타일에 따라 화면에 어떻게 나타나는지에 대한 규칙이 다릅니다. 예를 들어, full-screen 프리젠테이션 스타일은 항상 화면 전체를 포함합니다. 뷰 컨트롤러가 화면에 표시될 때, UIKit은 프리젠테이션을 위한 적절한 context를 가진 뷰 컨트롤러를 찾습니다. 많은 경우에 UIKit은 가장 가까이 있는 뷰 컨트롤러를 선택하지만 윈도우의 루트 뷰 컨트롤러를 선택할 수도 있습니다. 몇몇의 경우 UIKit에게 어떤 뷰 컨트롤러가 프리젠테이션 context를 제공하고 처리할 수 있는지 알릴 수도 있습니다.

Figure 2-4 shows why containers usually provide the context for a presentation. When performing a full-screen presentation, the new view controller needs to cover the entire screen. Rather than requiring the child to know the bounds of its container, the container decides whether to handle the presentation. Because the navigation controller in the example covers the entire screen, it acts as the presenting view controller and initiates the presentation.

그림 2-4는 컨테이너가 프리젠테이션을 위한 context를 제공해야 하는지 보여줍니다. full-screen 프리젠테이션이 수행될 때, 새로운 뷰 컨트롤러는 전체 화면을 커버해야 합니다. 자식이 컨테이너의 크기, 경계를 알게 하는 것보다 컨테이너가 프리젠테이션을 처리할 지 결정하는 것이 좋습니다. 아래 그림의 네비게이션 컨트롤러는 전체 화면을 커버하기 때문에, presenting 뷰 컨트롤러로 동작하고 프리젠테이션을 시작합니다.

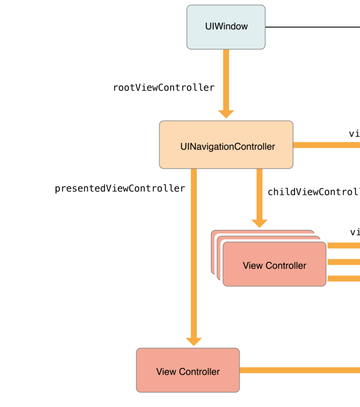

Figure 2-4 A container and a presented view controller

For information about presentations, see The Presentation and Transition Process.

프리젠테이션에 대한 자세한 정보는 The Presentation and Transition Process 를 참고하세요.