Understanding Responders and the Responder Chain

Apps receive and handle events using responder objects. A responder object is any instance of the `UIResponder` class, and common subclasses include `UIView`, `UIViewController`, and `UIApplication`. UIKit manages most responder-related behavior automatically, including how events are delivered to one responder to the next. However, you can modify the default behavior to change how events are delivered within your app.

앱은 리스폰더 객체 들을 이용하여 이벤트를 수신하고 처리합니다. 리스폰더 객체는 `UIResponder`클래스의 서브 클래스이며 일반적으로 `UIView`, `UIViewController`, `UIApplication`의 서브클래스 입니다. UIKit은 이벤트가 어떻게 하나의 리스폰더에서 다음으로 전달되는 방법을 포함한 대부분의 응답과 관련된 행동을 자동으로 관리합니다. 그러나 여러분은 이 기능을 수정하여 여러분의 앱에서 전달되는 이벤트가 전달되는 방식을 변경할 수 있습니다.

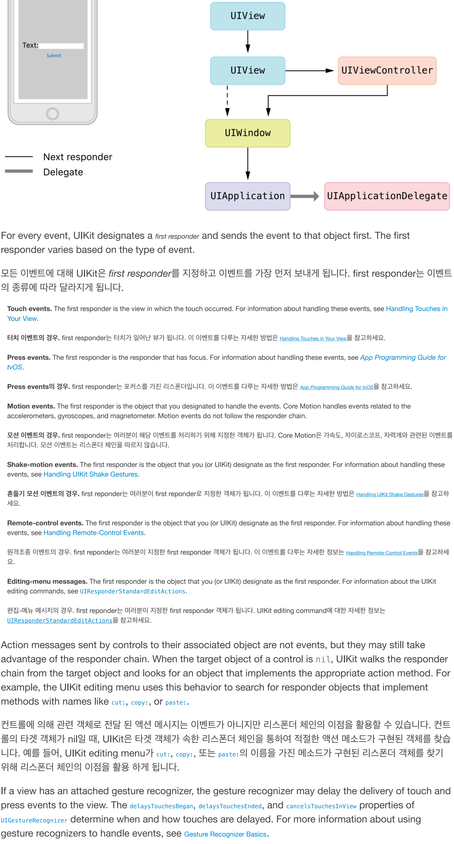
UIKit directs most events to the most appropriate responder object in your app. If that object does not handle the event, UIKit forwards it to the next responder in the active *responder chain*, which is a dynamic configuration of your app's responder objects. Because it is dynamic, there is no single responder chain within your app. However, it is easy to determine the next responder in the chain because events always flow from specific responder objects to more general responder objects. For example, the next responder for a view is either its superview or the view controller that manages it. Events continue to flow up the responder chain until they are handled.

UIKit은 대부분의 이벤트를 가장 적절한 리스폰더 객체로 전달합니다. 만약 그 객체가 해당 이벤트를 처리하지 않는 객체라면, UIKit은 앱의 리스폰더 객체 들을 동적으로 구성하는 활성화 된 *responder chain*의 다음 리스폰더 객체로 해당 이벤트를 전달하게 됩니다. 동적이기 때문에 리스폰더 체인은 단일로 구성되진 않습니다. 그러나 이벤트는 항상 특정 리스폰더 객체에서 좀더 일반적인 리스폰더 객체로 흐르기 때문에 다음 리스폰더를 리스폰더 체인에서 좀 더 쉽게 결정할 수 있습니다. 예를 들어, 한 뷰의 다음 리스폰더 객체가 해당 객체의 부모 뷰 이거나 뷰 컨트롤러라면 이벤트는 처리될 때까지 리스폰더 체인 워로 계속해서 흘러가게 됩니다.

Figure 7-1 shows the responder chains that form in an app whose interface contains a label, a text field, a button, and two background views. If the text field does not handle an event, UIKit sends the event to the text field's parent `UIView` object, followed by the root view of the window. From the root view, the responder chain diverts to the owning view controller before returning to the view's window. If the window does not handle the event, UIKit delivers the event to the `UIApplication` object, and possibly to the app delegate if that object is an instance of `UIResponder` and not already part of the responder chain.

아래의 그림 7-1에서는 레이블, 텍스트 필드, 버튼, 두 개의 백그라운드 뷰로 구성된 인터페이스 형식을 가진 리스폰더 체인을 보여주고 있습니다. 만약 텍스트 필드가 이벤트를 처리하지 않는다면, UIKit은 이벤트를 텍스트 필드의 부모 `UIView` 객체로 전달하고 그 다음에는 윈도우의 루트 뷰로 전달합니다. 루트 뷰에서 뷰의 윈도우로 반환되기 이전에 리스폰더 체인은 자신을 소유한 뷰 컨트롤러로 체인의 흐름을 바꾸게 됩니다. 만약 윈도우가 이벤트를 처리하지 않는다면, UIKit은 이벤트를 `UIApplication`객체에 전달하고 만약 해당 객체가 `UIResponder`의 인스턴스가 아니고 리스폰더 체인에 속하지 않는다면 앱 델리게이트에게 전달합니다.

Figure 7-1 An example of a responder chain



For every event, UIKit designates a *first responder* and sends the event to that object first. The first responder varies based on the type of event.

모든 이벤트에 대해 UIKit은 *first responder*를 지정하고 이벤트를 가장 먼저 보내게 됩니다. first responder는 이벤트의 종류에 따라 달라지게 됩니다.

Touch events. The first responder is the view in which the touch occurred. For information about handling these events, see Handling Touches in Your View.

터치 이벤트의 경우. first responder는 터치가 일어난 뷰가 됩니다. 이 이벤트를 다루는 자세한 방법은 Handling Touches in Your View을 참고하세요.

Press events. The first responder is the responder that has focus. For information about handling these events, see *App Programming Guide for tvOS*.

Press events의 경우. first responder는 포커스를 가진 리스폰더입니다. 이 이벤트를 다루는 자세한 방법은 *App Programming Guide for tvOS*을 참고하세요.

Motion events. The first responder is the object that you designated to handle the events. Core Motion handles events related to the accelerometers, gyroscopes, and magnetometer. Motion events do not follow the responder chain.

모션 이벤트의 경우. first responder는 여러분이 해당 이벤트를 처리하기 위해 지정한 객체가 됩니다. Core Motion은 가속도, 자이로스코프, 자력계와 관련된 이벤트를 처리합니다. 모션 이벤트는 리스폰더 체인을 따르지 않습니다.

Shake-motion events. The first responder is the object that you (or UIKit) designate as the first responder. For information about handling these events, see Handling UIKit Shake Gestures.

흔들기 모션 이벤트의 경우. first responder는 여러분이 first responder로 지정한 객체가 됩니다. 이 이벤트를 다루는 자세한 방법은 Handling UIKit Shake Gestures을 참고하세요.

Remote-control events. The first responder is the object that you (or UIKit) designate as the first responder. For information about handling these events, see Handling Remote-Control Events.

원격조종 이벤트의 경우. first responder는 여러분이 지정한 first responder 객체가 됩니다. 이 이벤트를 다루는 자세한 정보는 Handling Remote-Control Events을 참고하세요.

Editing-menu messages. The first responder is the object that you (or UIKit) designate as the first responder. For information about the UIKit editing commands, see `UIResponderStandardEditActions`.

편집-메뉴 메시지의 경우. first responder는 여러분이 지정한 first responder 객체가 됩니다. UIKit editing command에 대한 자세한 정보는 `UIResponderStandardEditActions`을 참고하세요.

Action messages sent by controls to their associated object are not events, but they may still take advantage of the responder chain. When the target object of a control is `nil`, UIKit walks the responder chain from the target object and looks for an object that implements the appropriate action method. For example, the UIKit editing menu uses this behavior to search for responder objects that implement methods with names like `cut:`, `copy:`, or `paste:`.

컨트롤에 의해 관련 객체로 전달 된 액션 메시지는 이벤트가 아니지만 리스폰더 체인의 이점을 활용할 수 있습니다. 컨트롤의 타겟 객체가 nil일 때, UIKit은 타겟 객체가 속한 리스폰더 체인을 통하여 적절한 액션 메소드가 구현된 객체를 찾습니다. 예를 들어, UIKit editing menu가 `cut:`, `copy:`, 또는 `paste:`의 이름을 가진 메소드가 구현된 리스폰더 객체를 찾기 위해 리스폰더 체인의 이점을 활용 하게 됩니다.

If a view has an attached gesture recognizer, the gesture recognizer may delay the delivery of touch and press events to the view. The `delaysTouchesBegan`, `delaysTouchesEnded`, and `cancelsTouchesInView` properties of `UIGestureRecognizer` determine when and how touches are delayed. For more information about using gesture recognizers to handle events, see Gesture Recognizer Basics.

만약 뷰가 제스쳐 recognizer를 가지고 있다면 제스쳐 recognizer는 해당 뷰로 향하는 터치 그리고 프레스 이벤트의 전달을 약간 늦추게 됩니다. `UIGestureRecognizer`의 프로퍼티 `delaysTouchesBegan`, `delaysTouchesEnded`, `cancelsTouchesInView`들은 어떻게 그리고 언제 터치가 딜레이 되는지 결정합니다. 이벤트를 처리하기 위해 gesture recognizer에 대한 자세한 정보는 Gesture Recognizer Basics를 참고하세요.

Determining Which Responder Contained a Touch Event

UIKit uses view-based hit testing to determine where touch events occur. Specifically, UIKit compares the touch location to the bounds of view objects in the view hierarchy. The `hitTest:withEvent:` method of `UIView` walks the view hierarchy, looking for the deepest subview that contains the specified touch. That view becomes the first responder for the touch event.

UIKit은 터치 이벤트가 일어난 곳을 찾기 위하여 뷰와 관련된 히트 테스팅을 이용합니다. 특히, UIKit은 뷰 계층안 의 뷰 객체의 영역과 터치가 일어난 위치를 비교합니다. `UIView`의 `hitTest:withEvent:`메소드는 뷰 계층을 돌며 특정 터치 영역을 포함하고 있는 가장 깊은 곳(사용자의 입장에선 가장 먼저 보이는)에 있는 서브 뷰를 찾습니다. 해당 뷰는 해당 터치 이벤트의 first responder가 되게 됩니다.

> NOTE
>
> If a touch location is outside of a view's bounds, the `hitTest:withEvent:` method ignores that view and all of its subviews. As a result, when a view's `clipsToBounds` property is NO, subviews outside of that view's bounds are not returned even if they happen to contain the touch. For more information about the hit testing behavior, see the discussion of the `hitTest:withEvent:` method in `UIView`.
>
> 만약 터치 영역이 해당 뷰의 영역 바깥에 있다면, `hitTest:withEvent:` 메소드는 해당 뷰의 그 뷰의 모든 서브뷰들을 무시하게 됩니다. 그 결과 뷰의 프로퍼티가 NO로 설정되었을 때, 뷰의 영역 바깥에 있는 서브 뷰들은 터치 영역 안에 있다고 하더라도 반환되지 못합니다. 히트 테스팅에 대한 자세한 정보는 `UIView`의 `hitTest:withEvent:`레퍼런스를 참고하세요.

UIKit permanently assigns each touch to the view that contains it. UIKit creates each `UITouch` object when the touch first occurs, and it releases that touch object only after the touch ends. As the touch location or other parameters change, UIKit updates the `UITouch` object with the new information. Several properties of the touch do not change, including the assigned view. Even when the touch location moves outside the original view, the value in the touch's `view` property remains the same.

UIKit은 각 터치를 포함하는 뷰에 영구적으로 각 터치를 할당 합니다. UIKit은 터치 이벤트가 처음 발생했을 때 `UITouch`객체를 생성하고 오직 터치가 끝난 후에만 릴리스 합니다. 터치 영역 또는 다른 파라메터의 변화가 있다면, UIKit은 `UITouch` 객체를 새로운 정보와 함께 업데이트합니다. 할당 된 뷰를 포함한 터치의 여러 프로퍼티 들은 변하지 않습니다. 터치 영역이 뷰의 바깥으로 이동하더라도 터치의 `view` 프로퍼티의 값은 동일하게 유지됩니다.

Altering the Responder Chain

You can alter the responder chain by overriding the `nextResponder` property of your responder objects. Many UIKit classes already override this property and return specific objects.

여러분은 리스폰더 객체의 `nextResponder` 프로퍼티를 재정의 함으로 리스폰더 체인을 조정할 수 있습니다. 많은 UIKit 클래스는 이미 이 프로퍼티를 재정의 하여 특정 객체를 반환하고 있습니다.

If you override the `nextResponder` property for any class, the next responder is the object you return.

만약 특정 클래스의 `nextResponder` 프로퍼티를 재정의 한다면, 다음 리스폰더는 여러분이 반환한 객체가 될 것입니다.

`UIView`

If the view is the root view of a view controller, the next responder is the view controller.

만약 해당 뷰가 뷰 컨트롤러의 루트 뷰라면, 다음 리스폰더는 뷰 컨트롤러가 됩니다.

If the view is not the root view of a view controller, the next responder is the view's superview.

만약 해당 뷰가 뷰 컨트롤러의 루트 뷰가 아니라면, 다음 리스폰더는 부모 뷰가 됩니다.

`UIViewController`

If the view controller's view is the root view of a window, the next responder is the window object.

만약 뷰 컨트롤러의 뷰가 윈도우의 루트 뷰라면, 다음 리스폰더는 윈도우 객체가 됩니다.

If the view controller was presented by another view controller, the next responder is the presenting view controller.

만약 뷰 컨트롤러가 다른 뷰 컨트롤러에 의해 표시된 뷰 컨트롤러라면, 다음 리스폰더는 presenting 뷰 컨트롤러가 됩니다.

`UIWindow`. The window's next responder is the application object.

윈도우의 다음 리스폰더는 어플리케이션 객체가 됩니다.

`UIApplication`. The app object's next responder is the app delegate, but only if the app delegate is an instance of `UIResponder` and is not a view, view controller, or the app object itself.

어플리케이션 객체의 다음 리스폰더는 앱 델리게이트가 되지만, 만약 앱 델리게이트가 클래스의 인스턴스이고 뷰, 뷰 컨트롤러 또는 어플리케이션 객체 그 자체가 아닐 경우에만 해당됩니다.

For more information about responders, see `UIResponder`.

리스폰더에 대한 자세한 정보는 `UIResponder`의 레퍼런스를 참고하세요.