iOS UIView绘制（二）View Hierarchy

发表于 2019-08-26 | 分类于 技术 | Valine： 0

本文主要围绕`UIView`从创建、添加到父视图、最后从父视图移除的相关过程，介绍`UIView`和`UIViewController`、`UIWindow`、以及`UIView`自身的关系。同是介绍生命周期，和上一篇有些重复，但上一篇主要是对`UIView`从初始化到渲染整体流程进行概括，这一篇仅仅围绕初始化进行，不涉及布局和渲染，并且结合demo分析了开发中常见的问题。

对象的创建

这里的`UIView`创建指的是我们需要添加到`UIViewController`中的视图，`UIViewController`的`Root View`，也就是我们常调用的`self.view`不需要我们创建，由`UIViewController`自动创建。

构造方法

创建一个`UIView`，第一步是调用构造方法，究竟调用哪个方法，取决于是从代码创建，还是使用了`Interface Builder`：

- 通过纯代码创建一个`UIView`对象，并初始化它的`frame`：

```
public init(frame: CGRect)
```

- 如果使用了xib或者storyboard，初始化就不会调用上面的方法，而是另一个方法：

```
public init?(coder aDecoder: NSCoder)
```

虽说两个方法都是“初始化”，但工作并不一样：通过纯代码创建一个`UIView`对象，就真真正正只是创建了一个对象。但使用了xib或者storyboard时，整个逆序列化的过程不仅仅是初始化对象，而是做了所有代码该做的事情，例如设置`UIView`相关属性、建立View树调整View层级等等。

自定义UIView

对于自定义的UIView，需要同时实现两个构造方法，所以需要初始化的相关操作可以这样写：

```
class TestView: UIView {

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupView()
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
                setupView()
    }

    private func setupView() {
                backgroundColor = .red
                }

    override func awakeFromNib() {
        super.awakeFromNib()
    }
}
```

另外，**不要使用awakeFromNib进行相关属性的初始化**，先看看苹果的官方文档：

> **Note**
>
> During Interface Builder's test mode, this message is also sent to objects instantiated from loaded Interface Builder plug-ins. Because plug-ins link against the framework containing the object definition code, Interface Builder is able to call their `awakeFromNib` method when present. The same is not true for custom objects that you create for your Xcode projects. Interface Builder knows only about the defined outlets and actions of those objects; it does not have access to the actual code for them.
>
> **Important**
>
> Because the order in which objects are instantiated from an archive is not guaranteed, your initialization methods should not send messages to other objects in the hierarchy. Messages to other objects can be sent safely from within an `awakeFromNib` method.

文档中明确说明，同一归档中的所有对象初始化完成之后才会调用`awakeFromNib`方法，并且对象初始化的顺序是不固定的，所以`awakeFromNib`的调用顺序和View的父子关系并不能对应，导致这种方式出现问题。

构建View Hierarchy

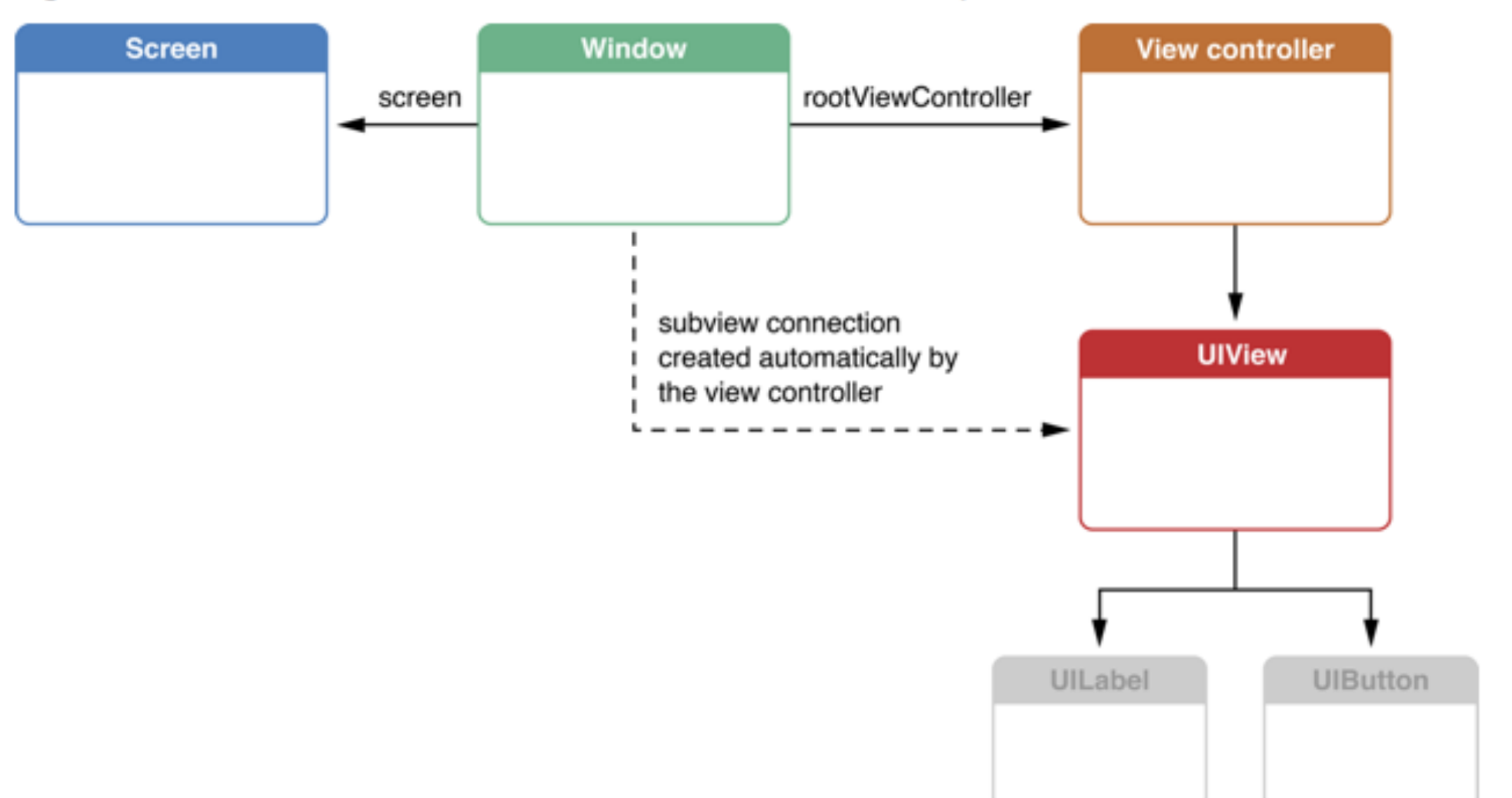
创建完`UIView`，设置完相关属性，一般下一步就是`self.view.addSubview()`。子View可以通过`superview`获取父View，父View可以通过`view.subviews`获取一个UIView数组，这个数组是按顺序排列的，越靠前的`UIView`越靠后（可以理解为Z轴越小）。`addSubview()`会把子View添加到数组末尾，因此如果有重叠的话，后面添加的View会遮盖之前的View。当然也可以通过`insertSubview`来直接选择插入的位置，可以指定index，或指定上下层的View。

这一步是在构建View树，也就是View的层级，这是`UIView`和其他`UIView`的关系。实际上`UIViewController`和`UIView`的关系，都是通过`UIViewController`的根视图，也就是我们常用的`self.view`关联起来的。`UIViewController`持有根视图，根视图又管理者所有子视图。除了响应链，一个`UIView`对于它所在的`UIViewController`是没有感知的。

联动UIWindow

** `UIWindow`是一种特殊的`UIView` **，iOS程序启动后创建的第一个视图就是`UIWindow`，然后是`UIViewController`的根`View`，接下来把这个根`View`添加为`UIWindow`的子`View`上。因为`self.view.addSubview`的操作，创建了View层级树，`UIViewController`包含的所有`View`也通过`Root View`和`UIWindow`有了联系。这个过程可以用下图来表示：

Figure 1-3 A view controller attached to a window automatically adds its view as a subview of the window



新创建一个iOS工程时，默认会执行以下操作：

1. 创建`UIWindow`：自动创建一个`UIWindow`，即自动创建的`AppDelegate`中的`window`属性
2. 创建`UIViewController`：自动创建`Main.storyboard`，将展示的第一个`UIViewController`指定为自动创建的`ViewController`，即`storyboard`中箭头指向的
3. 绑定：把上面创建的`ViewController`指定为`UIWindow`的`Root ViewController`，同时`Root ViewController`的`Root View`成为`UIWindow`的子`View`

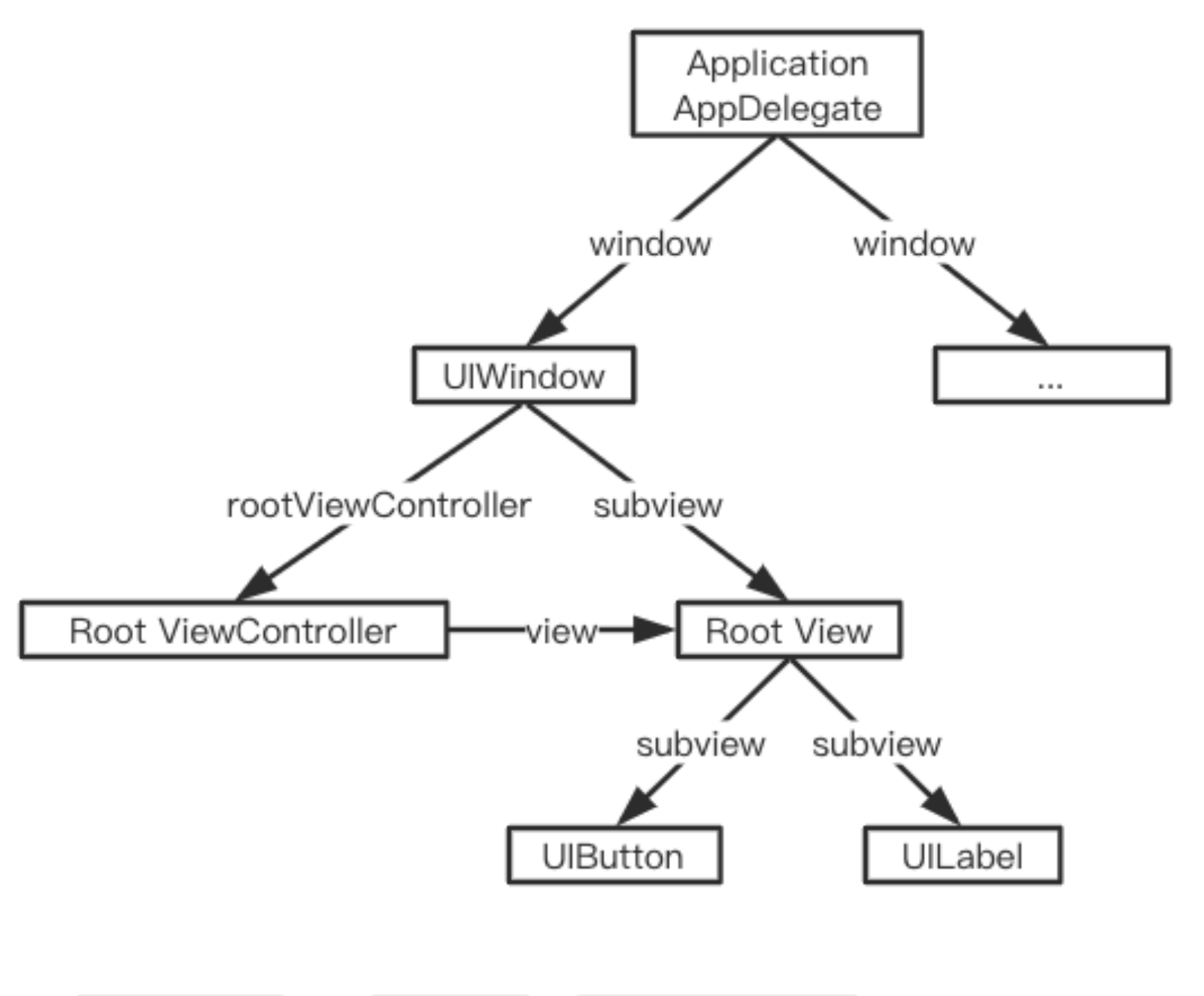
因为使用了`storyboard`，这个过程被隐藏了：从Main`函数执行`，到创建`Application`，到设置`AppDelegate`，再到执行上述步骤，最后调用`didFinishLaunchingWithOptions`。整个过程变得不可见，也方便了很多，实际上可以用代码这样表示：

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

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptio
        // Override point for customization after application launch.
        self.window = UIWindow(frame: UIScreen.main.bounds)
        self.window?.backgroundColor = .white
        let viewController = ViewController()
        self.window?.rootViewController = viewController
        self.window?.makeKeyAndVisible()
        return true
    }
}
```

为了便于理解这几个对象之间的关系，我画了一张图：



切记`UIWindow`也是`UIView`，`view.window`获取的就是View树的根节点。也就是说，`View`的添加可以不经过`ViewController`，但推荐使用上面这种方式：通过`self.window?.rootViewController = viewController`将`UIWindow`和`UIViewController`绑定起来。

一般情况下，不要使用另一种方式：`self.window?.addSubview(view)`。这种方式越过了`ViewController`，`view`不再被`ViewController`管理，而是直接和`window`发生关系，可能引发一些问题：

- 当view产生事件，通知控制器的时候，控制器可能已经销毁
- 发生旋转屏幕等事件时，事件传递：`UIApplication --> UIWindow --> Root ViewController --> View`，如果没有了`ViewController`，`View`就收不到消息，不能响应事件

总结

本文介绍了`UIView`创建的过程，以及这个过程中相关对象的行为，例如`UIViewController`和`UIWindow`，对应的方法也比较明了：

1. 对象的初始化：主要是构造方法`init`
2. 构建View树：父View的`addSubview`和子View的`willMove(toSuperview newSuperview: UIView?)`、`didMoveToSuperview`
3. 联动`UIWindow`：View的`willMove(toWindow newWindow: UIWindow?)`、`didMoveToWindow`

技术 # iOS # View

< 抖音实习周记（五） iOS UIView绘制（三）从Layout到Display >

昵称 邮箱

欢迎批评指正

提交

来发评论吧~

Powered By Valine
v1.4.16