UIKit基础（2）

这节课我会学到什么？

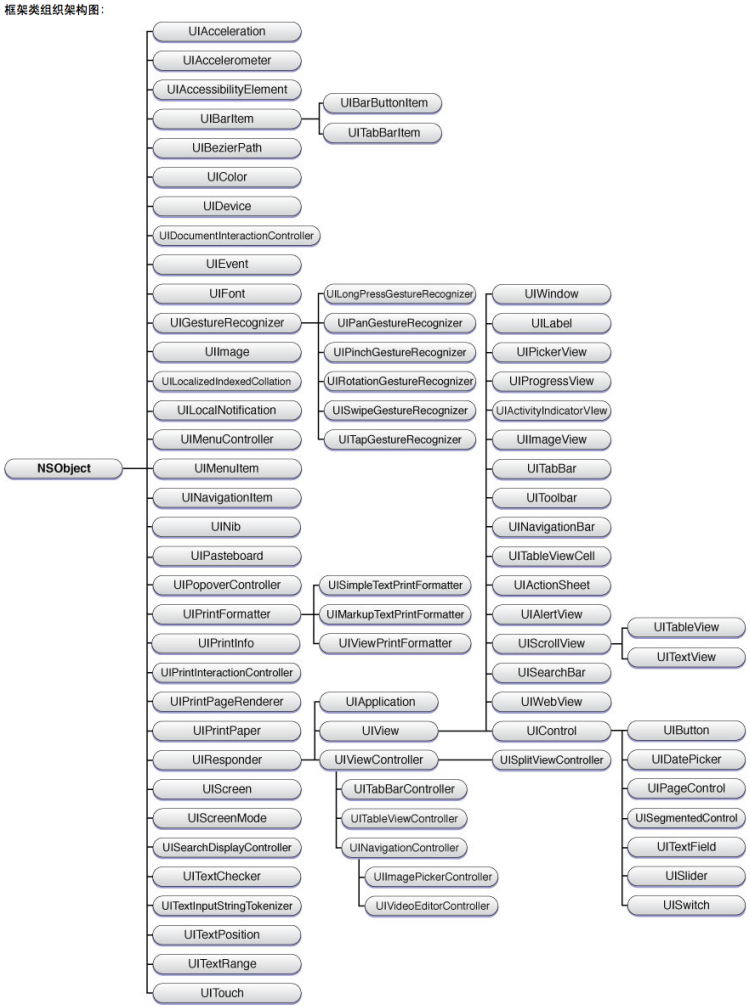
- UIView的常见属性
- UIViewController介绍
- UIWindow介绍
- UIApplication介绍

了解UIView

1. 能看得见的都是UIView，它是视图的基类，表示屏幕上的一块矩形区域
2. 每个UIView都可以容纳其他UIView（容器）

UIKit家族树

黑核科技 CORE TECH.



UIControl

- 活动控件，可以与用户交互
- UIControl是UIView 的子类
- UIControl是诸如UIButton、UITextField等控件的父类，它本身也包含了一些属性和方法，但是不能直接使用UIControl类，它只是定义了子类都需要使用的方法。

UIView的frame,bounds和center

frame：描述当前视图在其父视图中的位置和大小。（CGRect）

参照父亲的坐标系统（改变位置和大小）

bounds：描述当前视图在其自身坐标系统中的位置和大小。（CGRect）

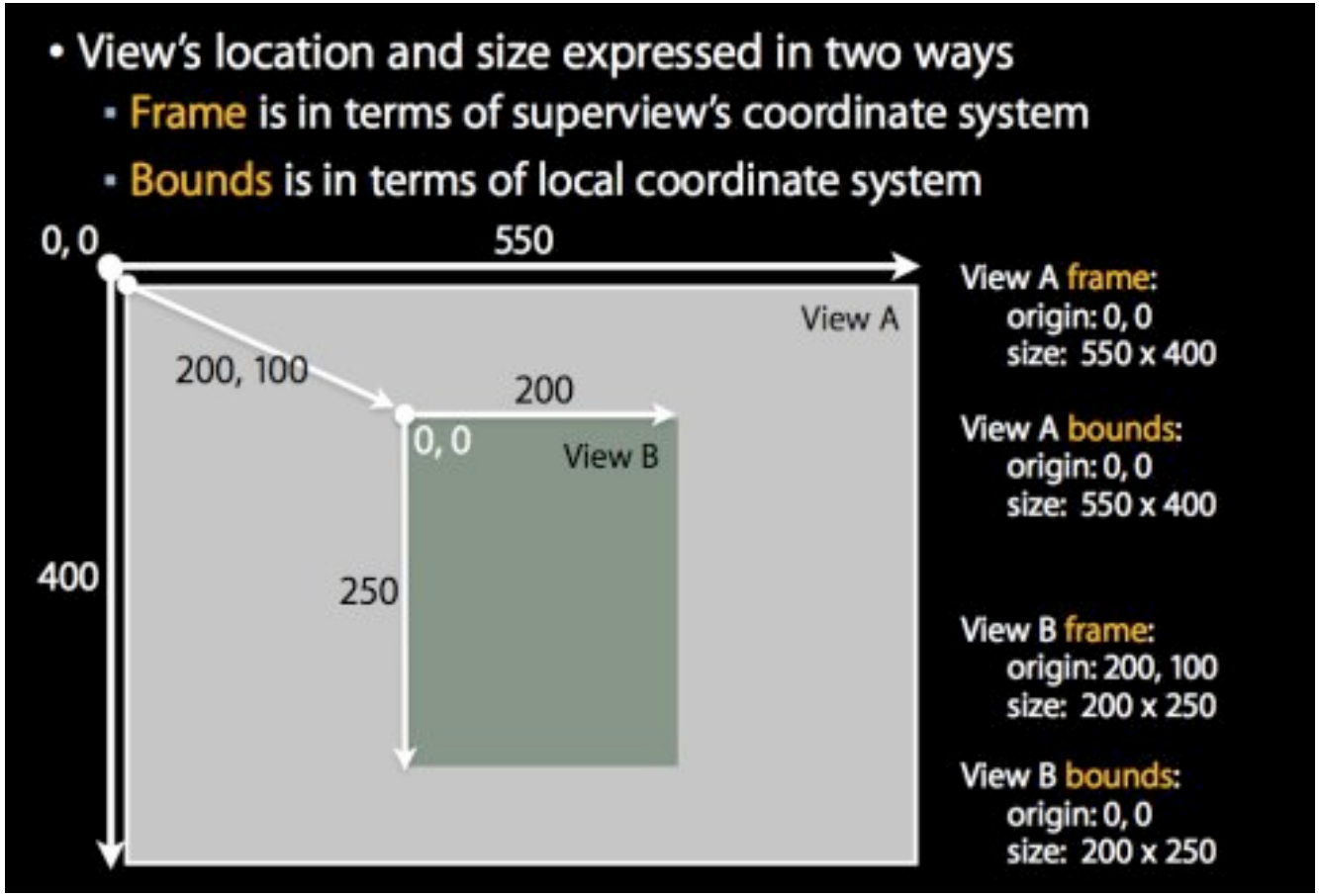
参照自己的坐标系统（改变大小）

center：描述当前视图的中心点在其父视图中的位置。（CGPoint）

参照父亲的坐标系统（改变位置）

iOS的坐标系

iOS以左上角为坐标原点(0,0)，以原点向右侧为X轴正方向，
原点下侧为Y轴正方向，如下图所示。



UIView的tag

黑核科技 CORE TECH.

- tag：必须唯一，指定一个View的tag后，将作为该控件的唯一标识。
- 注意：tag指定后不需要也不要去修改。
- 作用：1. 区分控件（如多个按钮调用同一个方法）
  2. 根据tag值查找视图

```
UILabel *label = (UILabel*)[self.view viewWithTag:101];
```

UIView的superview和subviews

- superview：父视图
- subviews： 子视图们
- 当一个视图B添加到视图A上时，视图A就成为视图B的父视图，视图B就成为视图A的子视图
- 一个视图可以有多个子视图（数组），但是只有一个父视图

调整UIView的层次结构常用方法

- addSubview: //添加子视图
- insertSubview:atIndex: //视图插入到指定索引位置
- insertSubview:aboveSubview: //视图插入指定视图之上
- insertSubview:belowSubview: //视图插入指定视图之下
- bringSubviewToFront: //把视图移动到最顶层
- sendSubviewToBack: //把视图移动到最底层
- exchangeSubviewAtIndex:withSubviewAtIndex //把两个索引对应的视图调换位置
- removeFromSuperview //把视图从父视图中移除

Alpha和opaque

alpha：透明度（0~1）如果将控件设为半透明效果，系统需要更多额外的计算开销来计算透明度，应尽量避免将UI控件设置成为半透明效果。如果全透明，可以直接使用hidden。

opaque：不透明，为YES时表示该控件是不透明的控件

如果你有没有透明度定义的视图，应该设置他们的opaque属性为YES。这会允许系统以最优的方式绘制你的views。 苹果的文档中有对这个属性的描述：这个属性提供了一个提示给图系统如何对待这个视图。如果设置为YES，绘制系统将会把这个视图视为完全不透明。这样允许系统优化一些绘制操作和提高性能（此视图后面的都不用绘制）。如果设置为NO，绘图系统会复合这个视图和其他的内容，这个属性的默认值是YES

UIView的其他常用属性

background：背景颜色

hidden：隐藏

User Interaction Enabled: 是否允许与用户交互（手指点击控件的时候，时候支持一定的响应）

UIView的形变属性

transform：形变属性

在OC中，通过transform属性可以修改对象的平移、缩放比例和旋转角度

常用的创建transform结构体方法分两大类

（1） 创建“基于控件初始位置”的形变

CGAffineTransformMakeTranslation（平移）

CGAffineTransformMakeScale（缩放）

CGAffineTransformMakeRotation（旋转）

（2） 创建“基于transform参数”的形变

CGAffineTransformTranslate

CGAffineTransformScale

CGAffineTransformRotate

UIViewController

1.视图控制器，顾名思义就是控制视图的，它可以方便地管理视图中的子视图。

2.一般情况下，每一个满屏的UiView都要交给一个UIViewController去管理（多视图多个UIViewController），每一个UIViewController都有一个UIView的属性。它是MVC开发模式里面的C（Controller）

3.作用：负责界面上元素（控件）的管理，包括创建、销毁，显示、隐藏，以及处理view和用户之间的交互（事件处理）等，是UIView的管家

4.逻辑：先创建一个UIViewController，再由UIViewController创建里面的UIView，然后把UIView展示到用户眼前，并且由UIViewController来处理UIVIew事件（如：点击）。

UIViewController的生命周期 *

- UIViewController有一些自带的方法，如loadView、viewDidLoad、viewWillAppear等，这些方法跟控制器的生命周期有着密切的联系。

UIViewController的生命周期 *

- 简单地讲，这些方法的作用以及调用顺序为：

```
- initWithNibName:bundle    // 初始化控制器
                            // (如果使用storyBoard则不执行该方法)
- loadView                  // 创建一个view赋值给控制器根视图
- viewDidLoad               // 进一步初始化
- viewWillAppear:           // 视图将要显示在屏幕上
- viewDidAppear:            // 视图已经在屏幕上渲染完成
- viewWillDisappear:        // 视图要被移除
- viewDidDisappear:         // 视图已被移除（屏幕上看不见了）
- dealloc                   // 视图已经销毁
```

UIViewController的生命周期 *

1.初始化方法:initWithNibName:bundle:（非必须）

初始化UIViewController，执行关键数据初始化操作，注意这里不要做view相关操作，view在loadView方法中才初始化，这时 loadView还未调用。

**如果使用StoryBoard进行视图管理，程序不会直接初始化一个UIViewController，StoryBoard 会自动初始化或在segue被触发时自动初始化，因此方法initWithNibName:bundle:不会被调用。

2.当访问UIViewController的view属性时，view如果此时是nil，那么VC会自动调用loadView方法来初始化一个UIView 并赋值给view属性（这个view可以是自己创建的（重写父类方法），也可以是调用父类创建的（[super loadView]））。需要注意的是，在view初始化之前，不能先调用view的getter方法，否则将导致死循环（除非先调用了[supper loadView];）。

UIViewController的生命周期 *

3.当VC的view对象载入内存后紧接着就会调用VC的viewDidLoad方法，这个时候VC.view保证是有值的，可以做进一步的初始化操作，例如添加一些subview。

4.在view即将添加到视图层级中(显示给用户)且任意显示动画切换之前调用viewWillAppear。这个方法中完成任何与视图显示相关的任务，例如改变视图方向、状态栏方向、视图显示样式等

5. viewDidAppear:在view被添加到视图层级中，显示动画切换之后调用。

UIViewController的生命周期 *

8. viewWillDisappear

view即将从superView中移除且移除动画切换之前，此时还没有调用removeFromSuperview。

9. viewDidDisappear

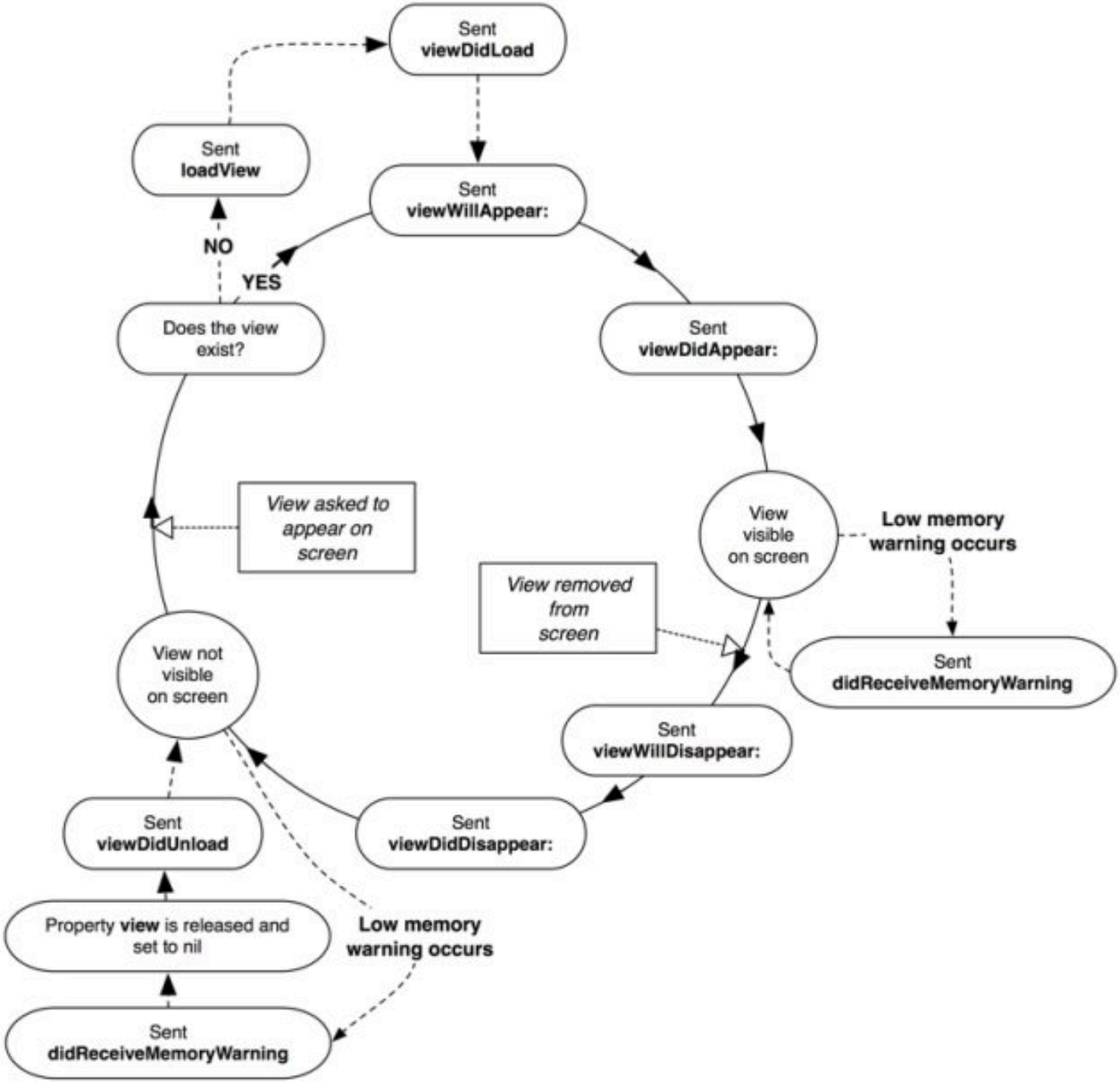
view从superView中移除，移除动画切换之后调用，此时已调用removeFromSuperview。

10.dealloc:释放资源(ARC可以忽略)

11. didReceiveMemoryWarning

当收到MemoryWarning的时候会卸载一些不必要的对象

**这个方法没有固定的调用时间，而是在设备出现内存警告的时候触发


Sent
viewDidLoad
Sent
loadView
Sent
viewWillAppear:
NO
YES
Does the view
exist?
Sent
viewDidAppear:
View asked to
appear on
screen
View
visible
on screen
Low memory
warning occurs
View removed
from
screen
View not
visible
on screen
Sent
didReceiveMemoryWarning
Sent
viewWillDisappear:
Sent
viewDidUnload
Sent
viewDidDisappear:
Property view is released and
set to nil
Low memory
warning occurs
Sent
didReceiveMemoryWarning

UIApplication—app大总管

1、UIApplication对象是应用程序的象征，一个UIApplication对象就代表一个应用程序。

2、每一个应用都有自己的UIApplication对象，而且是单例的，通过[UIApplication sharedApplication]可以获得这个单例对象

3、 一个iOS程序启动后创建的第一个对象就是UIApplication对象，且只有一个。

很多时候，我们不需要关心这个类，我们很少继承这个类，偶尔会调用这个类的方法来实现一些功能，但是不可否认，这个类是iOS编程中很重要的一个概念。UIApplication的核心作用是提供了iOS程序运行期间的控制和协作工作。

UIApplication常用属性

利用UIApplication对象，能进行一些应用级别的操作：

1.网络活动指示器（处于状态栏上的小菊花）

[UIApplication sharedApplication] . networkActivityIndicatorVisible

2.打开URL

[UIApplication sharedApplication] .openURL:

3.状态栏风格

[UIApplication sharedApplication] .statusBarStyle

4.状态栏是否可见

[UIApplication sharedApplication] .statusBarHidden

5.获取状态栏方向

[UIApplication sharedApplication] .statusBarOrientation

6.获取状态栏尺寸

[UIApplication sharedApplication] .statusBarFrame

7.设置push通知作用到应用程序图标上的红色的圈圈数字

[UIApplication sharedApplication] .applicationIconBadgeNumber

UIApplication Delegate

app很容易受到比如来电的干扰，受到干扰时，会产生一些系统事件，这时UIApplication会通知它的delegate对象，让delegate代理来处理这些系统事件。

新建的工程自带的AppDelegate类，这个类已经默认遵守了UIApplicationDelegate协议，成为UIApplication的代理

AppDelegate的代理方法

// 当应用程序启动完毕的时候就会调用(系统自动调用)

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions

// 即将失去活动状态的时候调用(失去焦点, 不可交互)

- (void)applicationWillResignActive:(UIApplication *)application

// 重新获取焦点(能够和用户交互)

- (void)applicationDidBecomeActive:(UIApplication *)application

// 应用程序进入后台的时候调用

// 一般在该方法中保存应用程序的数据, 以及状态

- (void)applicationDidEnterBackground:(UIApplication *)application

// 应用程序即将进入前台的时候调用（一般在该方法中恢复应用程序的数据,以及状态）

- (void)applicationWillEnterForeground:(UIApplication *)application

// 应用程序即将被销毁的时候会调用该方法（如果应用程序处于挂起状态的时候无法调用该方法）

- (void)applicationWillTerminate:(UIApplication *)application

AppDelegate的代理方法

didFinishLaunchingWithOptions	当应用程序启动完毕的时候调用（只调用一次）
applicationWillResignActive	失去焦点
applicationDidBecomeActive	重新获得焦点
applicationDidEnterBackground	进入后台时候调用
applicationDidEnterBackground	进入前台时候调用
applicationWillTerminate	程序即将被销毁的时候调用

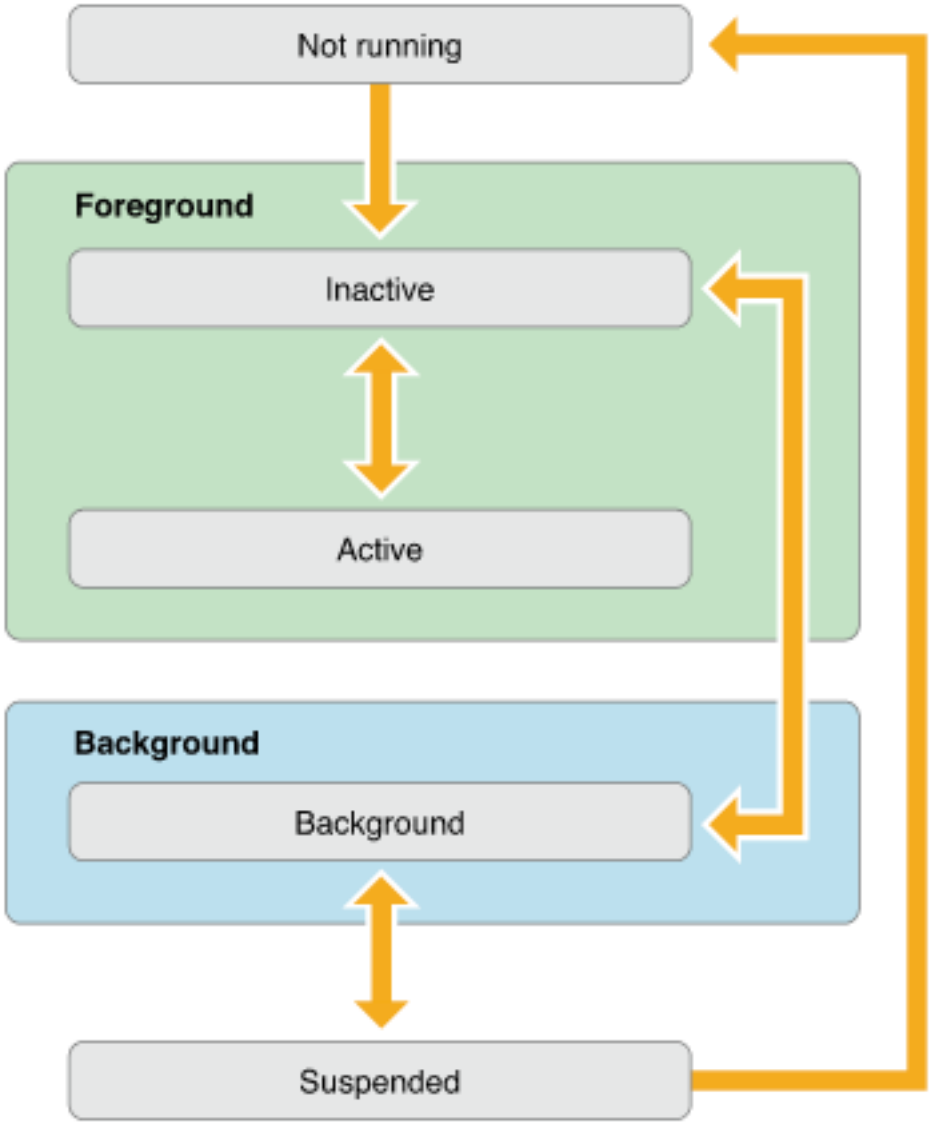
获取代理对象的方法


```
[[UIApplication sharedApplication] delegate];
```

应用程序的状态

iOS应用程序一共有五种状态：
1. Not running:应用还没有启动，或者应用正在运行但是途中被系统停止。
2. Inactive:当前应用正在前台运行，但是并不接收事件（当前或许正在执行其它代码）。一般每当应用要从一个状态切换到另一个不同的状态时，中途过渡 会短暂停留在此状态。唯一在此状态停留时间比较长的情况是：当用户锁屏时，或者系统提示用户去响应某些（诸如电话来电、有未读短信等）事件的时候。
3. Active:当前应用正在前台运行，并且接收事件。这是应用正在前台运行时所处的正常状态。
4. Background:应用处在后台，并且还在执行代码。大多数将要进入Suspended状态的应用，会先短暂进入此状态。然而，对于请求需要额外的执行时间的应用，会在此状态保持更长一段时间（如音乐播放器）。
5. Suspended:应用处在后台，并且已停止执行代码。系统自动的将应用移入此状态，且在此举之前不会对应用做任何通知。当处在此状态时，应用依然驻留内存但不执行任何程序代码。当系统发生低内存告警时，系统将会将处于Suspended状态的应用清除出内存以为正在前台运行的应用提供足够的内存。

应用程序的状态


Not running
Foreground
Inactive
Active
Background
Background
Suspended

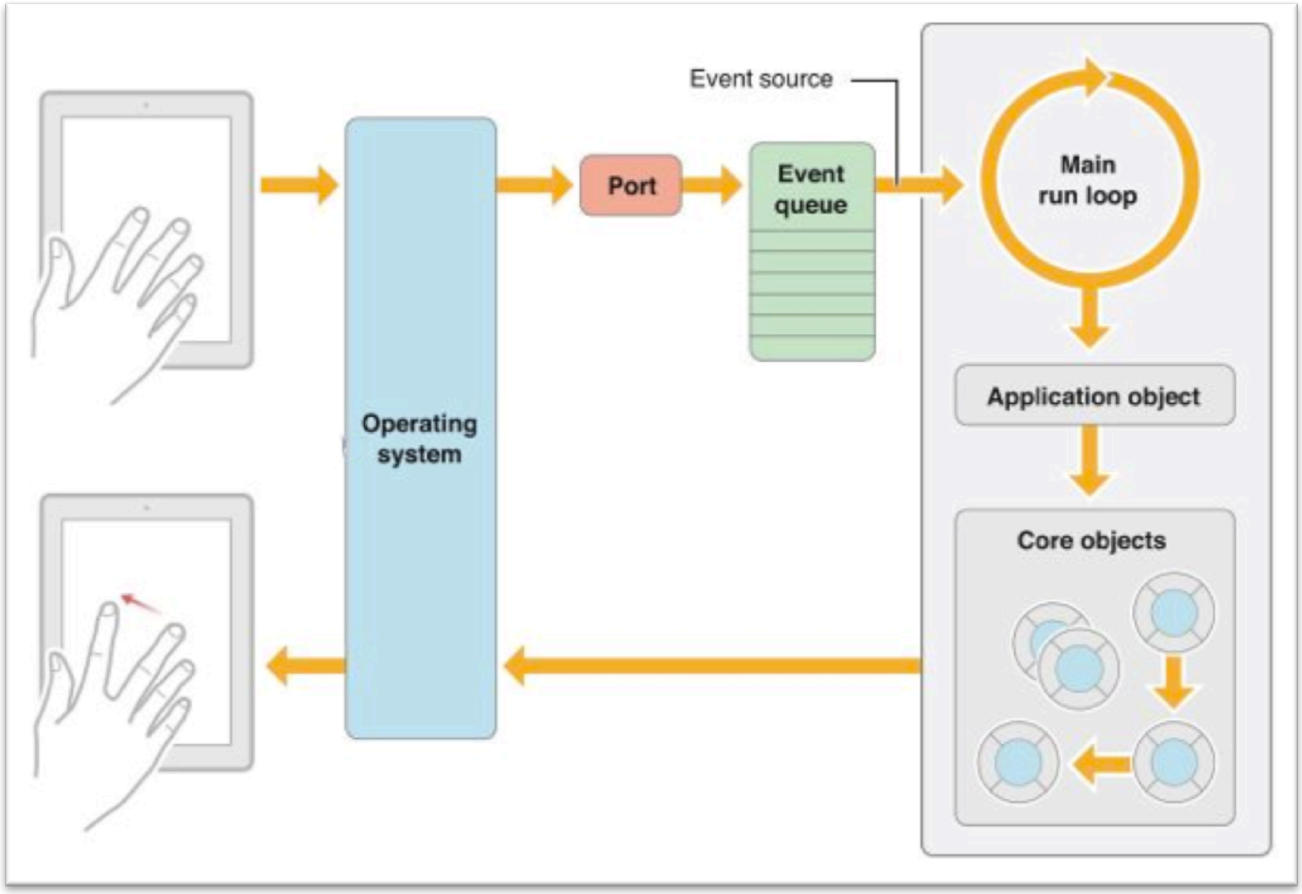
UIApplicationMain函数

在程序的入口main.m文件的main函数中，调用了一个叫做UIApplicationMain的函数。

```
9  #import <UIKit/UIKit.h>
10 #import "AppDelegate.h"
11
12 int main(int argc, char * argv[]) {
13     @autoreleasepool {
14         return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
15     }
16 }
```

argc 和argv是ISO C标准的main函数的参数，直接传递给UIApplicationMain进行相关处理（不用管）
第三个参数确定了主要应用程序类的名称，该类必须是UIApplication(或子类)，如果传值为nil，UIKit就会使用默认的程序类UIApplication。
第四个参数是程序自定义的代理类名，这个类负责系统和代码之间的交互，该类必须遵守UIApplicationDelegate协议。它一般在Xcode新建项目时会自动生成。

- 也就是说：UIApplicationMain函数会根据principalClassName（第三个参数）创建UIApplication对象，根据 delegateClassName（第四个参数）创建一个delegate对象，并将该delegate对象赋值给UIApplication对象中的delegate属性
- 接着会建立应用程序的Main Runloop（主运行循环），进行事件的处理



果核科技 CORE TECH.

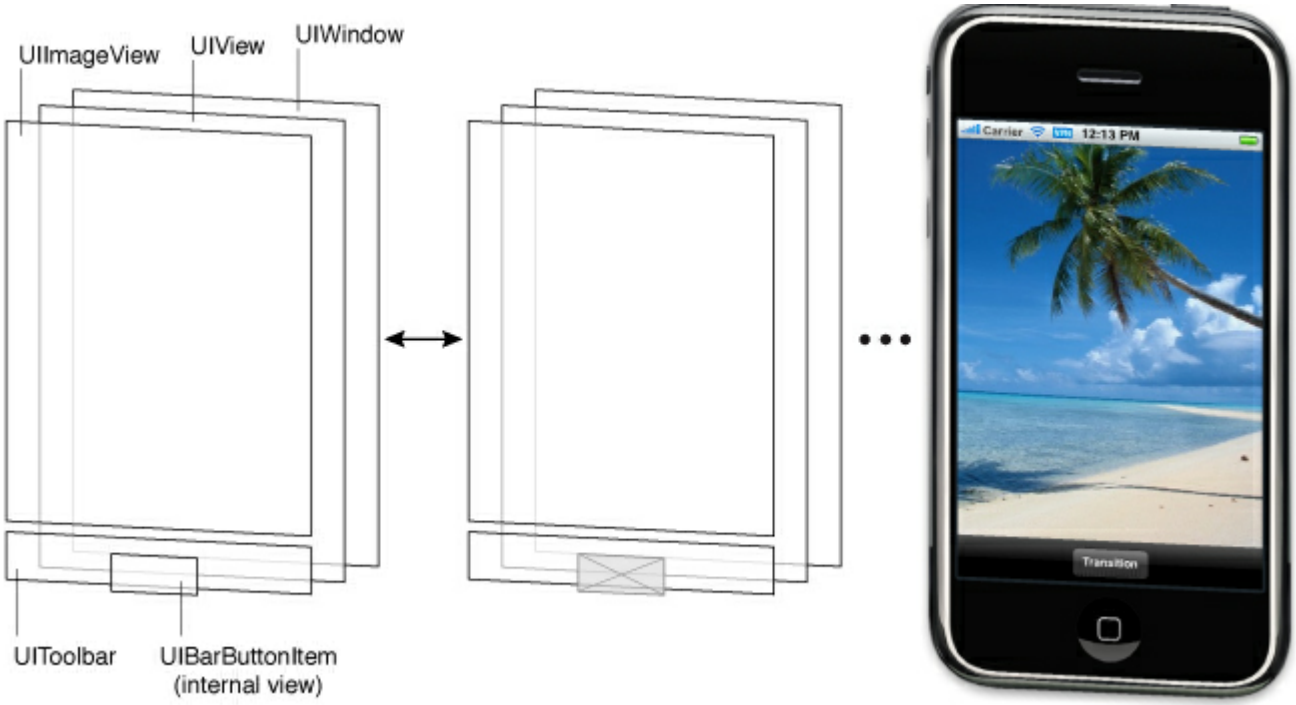
main Runloop怎么工作？

用户操作设备，相关的操作事件被系统生成并通过UIKit的指定端口分发。事件在内部排成队列，一个个的分发到Main runloop 去做处理。UIApplication对象是第一个接收到事件的对象，它决定事件如何被处理。触摸事件分发到主窗口，窗口再分发到对应出发触摸事件的 View。其他的事件通过其他途径分发给其他对象变量做处理。

UIWindow---App的大舞台

黑核科技 CORE TECH.

UIWindow是一种特殊的UIView，iOS程序启动完毕后，创建的第一个视图控件就是UIWindow（创建的第一个对象是UIapplication），再创建控制器，接着创建控制器的view，最后将控制器的view添加到UIWindow上，于是控制器的view就显示在屏幕上了。



UIWindow在程序中主要起到三个作用：

1、提供一个区域来显示UIView,一个iOS程序之所以能显示到屏幕上，完全是因为它有UIWindow。也就说，没有UIWindow，就看不见任何UI界面。

2、传递触摸消息到程序中view和其他对象

3、与UIViewController协同工作，方便完成设备方向旋转的支持

Tips:

- 一般情况下，应用程序只有一个UIWindow对象，即使有多个UIWindow对象，也只有一个UIWindow可以接受到用户的触屏事件。
- 不要把UIView直接添加到UIWIndow上面，虽然这样做是不会报错的，但是非常难于管理，苹果也不推荐我们这么做，而是通过添加一个UIViewController来添加视图。
- UIWindow可以通过设置根视图控制器，将控制器的视图添加的window上

UIWindow的创建过程(代码1)

第一步：创建UIWindow并显示

```
// 程序启动完成调用
-   (BOOL)application:(UIApplication *)application
 didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

   // 1.初始化一个屏幕的大小的UIWindow
      self.window = [[UIWindow alloc]initWithFrame:[UIScreen mainScreen].bounds];
   // 2.设置window的背景颜色
      self.window.backgroundColor = [UIColor whiteColor];
   // 3.设置为主窗口并显示出来
       [self.window makeKeyAndVisible];
   return YES;
}
```

如何创建一个不需要storyBoard的工程：

1.新建一个singleViewApplication

2.删除storyBoard文件

3.Info.plist里面删除Main storyboard file base name

4.在appDelegate的didFinishLaunchingWithOptions:方法里面写UIWindow的初始化方法

UIWindow的创建过程(代码2)

第二步：把view添加到uiwindow

1.创建要添加的view对应的ViewController

2.设置UIWindow的根控制器，UIWindow会自动将rootviewcontroller的view添加到window中，负责管理rootviewcontroller的生命周期

[self.window.rootviewcontroller=vc];

```
#import "AppDelegate.h"
#import "ViewController.h"

@interface AppDelegate ()

@end

@implementation AppDelegate

// 程序启动完成调用
- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // 1.初始化一个屏幕的大小的UIWindow
    self.window = [[UIWindow alloc]initWithFrame:[UIScreen mainScreen].
        bounds];

    // 创建一个控制器并作为self.window的根视图
    ViewController *vc = [[ViewController alloc]init];
    self.window.rootViewController = vc;

    // 2.设置window的背景颜色
    self.window.backgroundColor = [UIColor whiteColor];
    // 3.设置为主窗口并显示出来
    [self.window makeKeyAndVisible];
    return YES;
}
```

在有storyboard的项目中UIWindow是如何创建的？

为什么创建一个storyboard，没有看到创建uiwindow的过程？

它其实是把创建UIWindow的过程给屏蔽起来了。可以把代理的UIWindow的属性的值打印出来NSLog(@“window=%p”,self.window);

打印出来确实是有值的，说明确实创建了 UIWindow.不仅创建了UIWindow，默认还创建了UIWindow对应的控制器，也可以打印进行查看。

NSLog(@“%@“,self.window.rootviewcontroller);

有storyboard的项目中UIWindow的创建过程：

1.当用户点击应用程序图标的时候，先执行Main函数，执行UIApplicationMain（ ）,根据其第三个和第四个参数创建Application，创建代理，并且把代理设置给application，开启一个事件循环。

2.当程序加载完毕，会调用代理的didFinishLaunchingWithOptions:方法。在调用 didFinishLaunchingWithOptions:方法之前，会加载storyboard（项目配置文件info.plist里面的storyboard的name，根据这个name找到对应的storyboard），在加载的时候创建一个window，接下来会创建箭头所指向的控制器，把该控制器设置为UIWindow的根控制器，接下来再将window显示出来，即看到了运行后显示的界面。（关于这部分可以 查看storyboard的初始化的文档）

获取UIWindow

（1）[UIApplication sharedApplication].windows 在本应用中打开的UIWindow列表，这样就可以接触应用中的任何一个UIView对象

（2）[UIApplication sharedApplication].keyWindow（获取应用程序的主窗口）用来接收键盘以及非触摸类的消息事件的UIWindow，而且程序中每个时刻只能有一个UIWindow是keyWindow。

（3）view.window获得某个UIView所在的UIWindow

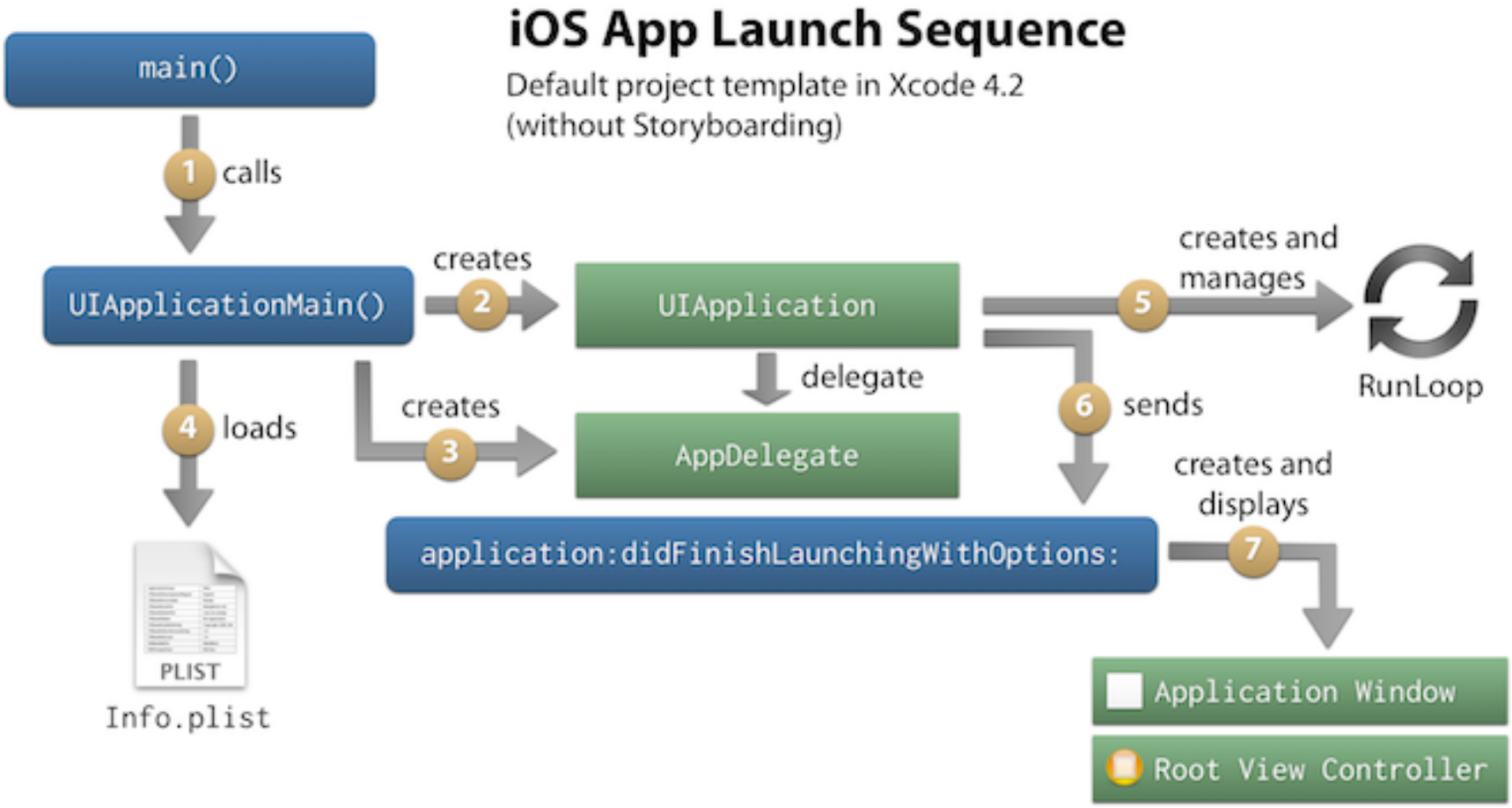
App启动过程

黑核科技 CORE TECH.

没有storyboard：
1.调用app的Main函数，执行UIApplicationMain（ ）
2. UIApplicationMain（ ）Application和代理，并且把代理设置给application
3.开启一个事件循环
4.当程序加载完毕，调用代理的 didFinishLaunchingWithOptions:方法。
5.在该方法中，创建一个Window,然后创建一个控制器，并把该控制器设置为UIWindow的根控制器，接下来再将window显示出来，即看到了运行后显示的界面。

有storyboard：
1.调用app的Main函数，执行UIApplicationMain（ ）
2. UIApplicationMain（ ）Application和代理，并且把代理设置给application
3.根据项目配置文件info.plist里面的 storyboard的name，找到对应的storyboard
4.接下来创建一个window，之后创建它的初始化控制器（就是箭头所指向的控制器）， 自动把该控制器设置为UIWindow的根控制器，接下来再将window显示出来，即看到了运行后显示的界面。

App启动过程


iOS App Launch Sequence
Default project template in Xcode 4.2
(without Storyboarding)
main()
1 calls
UIApplicationMain()
creates
2
UIApplication
creates and
manages
5
RunLoop
delegate
4 loads
creates
3
AppDelegate
6 sends
creates and
displays
7
application:didFinishLaunchingWithOptions:
PLIST
Info.plist
Application Window
Root View Controller

插播：代码编辑小技巧

1. 将书签添加到没个Xcode窗口顶端的方法列表弹出窗口中，可以利用#pragma mark组织源代码。

2. 折叠方法

作业

1. 理解并掌握UIViewController的生命周期
2. 理解并掌握app启动的过程
3. 自己创建一个无storyboard文件的项目，并实现界面的部分控件初始化。