# 框架

* 1. 框架概述

FrameWork是一些类的集合,通常是一些可能一起使用的类集合.换句话说,这些类被一起编译成可复用的代码库（library of code）。所有相关的资源和代码库一起放入一个带扩展名“.framework”的目录(Mac OSX bundle 的概念可以参考苹果的一些文档). 系统自带的framework可以在/System/Library/Frameworks找到

系统会在需要的时候将 framework 载入内存中，多个应用程序可以同时使用同一个 framework，而内存中的拷贝只有一份。一个 framework 同时也是一个 bundle，我们可以在 finder 里浏览其内容，也可以在代码中通过 NSBundle 访问它。利用 framework 我们可以实现动态或静态库的功能。与动态/静态库相比，framework 有如下优势：

1. framework 能将不同类型的资源打包在一起，使之易于安装，卸载与定位；  
   第二，framework 能够进行版本管理，这使得 framework 能不断更新并向后兼容；  
   第三，在同一时间，即使有多个应用程序使用同一 framework，但在内存中只有一份 framework 只读资源的拷贝，这减少了对内存的占用
   1. Cocoa

Cocoa有3个framework组成

* 1. Foundation: 所有的面向对象语言都会有一些标准结构: value , collection , String, dates, lists, thread等等.所有的这些都在Foundation framework里面(可以对比C++的STL)
  2. Appkit: 所有和用户界面相关的类都在这里. Windows,buttons,text field, event, drawing. 它还有个名字: ApplicationKit
  3. Core Data: Core Data可以方便的让你把你的对象存储成文件,或是从文件中加载你的对象
  4. CoreFoundation
     1. Core Foundation 框架

Core Foundation框架 (CoreFoundation.framework) 是一组C语言接口，它们为iOS应用程序提供基本数据管理和服务功能。下面列举该框架支持进行管理的数据以及可提供的服务：

群体数据类型 (数组、集合等)  
程序包  
字符串管理  
日期和时间管理  
原始数据块管理  
偏好管理  
URL及数据流操作  
线程和RunLoop  
端口和soket通讯

* + 1. Core Foundation和Foundation

二者紧密相关，它们为相同功能提供接口，但Foundation框架提供Objective-C接口。

如果您将Foundation对象和Core Foundation类型掺杂使用，则可利用两个框架之间的 “toll-free bridging”。**所谓的Toll-free bridging是说您可以在某个框架的方法或函数同时使用Core Foundatio和Foundation 框架中的某些类型。很多数据类型支持这一特性，其中包括群体和字符串数据类型。同时也要注意在没有ARC和xcode4.2后加入ARC的使用时的区别（使用这些关键字的原因就是引入ARC后一些转换会失败）**

在iOS世界，主要有两种对象：Objective-C 对象和 Core Foundation 对象0。Core Foundation 对象主要是有C语言实现的 Core Foundation Framework 的对象，其中也有对象引用计数的概念，只是不是 Cocoa Framework::Foundation Framework 的 retain/release，而是自身的 CFRetain/CFRelease 接口。

这两种对象间可以互相转换和操作，不使用ARC的时候，单纯的用C原因的类型转换，不需要消耗CPU的资源，所以叫做 Toll-Free bridged。比如 NSArray和CFArrayRef, NSString和CFStringRef，他们虽然属于不同的 Framework，但是具有相同的对象结构，所以可以用标准C的类型转换。

但是在ARC有效的情况下，将出现类似下面的编译错误，这个时候需要相应的桥接关键字

类型转换关键字区别：

\_\_bridge （相当于assign 还是要自己管理对象），\_\_bridge 转换类型和使用 \_\_unsafe\_unretained 关键字修饰的变量是一样的。被代入对象的所有者需要明确对象生命周期的管理，不要出现异常访问的问题

\_\_bridge\_transfer（相当于copy后release   转让了自己对对象的所有权交给别人去管理）  
\_\_bridge\_retained（相当于retain 增加额对象的引用计数），类型被转换时，其对象的所有权也将被变换后变量所持有。

如果不是ARC代码，类似下面的实现：

id obj = [[NSObject alloc] init];  
   
void \*p = obj;  
[(id)p retain];  
可以用一个实际的例子验证，对象所有权是否被持有。

void \*p = 0;  
   
{  
    id obj = [[NSObject alloc] init];  
    p = (\_\_bridge\_retained void \*)obj;  
}  
   
NSLog(@"class=%@", [(\_\_bridge id)p class]);  
出了大括号的范围后，p 仍然指向一个有效的实体。说明他拥有该对象的所有权，该对象没有因为出其定义范围而被销毁。

总结：

明确被转换类型是否是 ARC 管理的对象  
Core Foundation 对象类型不在 ARC 管理范畴内  
Cocoa Framework::Foundation 对象类型（即一般使用到的Objectie-C对象类型）在 ARC 的管理范畴内  
如果不在 ARC 管理范畴内的对象，那么要清楚 release 的责任应该是谁  
各种对象的生命周期是怎样的  
1. 声明 id obj 的时候，其实是缺省的申明了一个 \_\_strong 修饰的变量，所以编译器自动地加入了 retain 的处理，所以说 \_\_bridge\_transfer 关键字只为我们做了 release 处理。  
**(\_\_bridge T) op**：告诉编译器在 bridge 的时候不要做任何事情

**(\_\_bridge\_retained T) op**：（ ObjC 转 CF 的时候使用）告诉编译器在 bridge 的时候 retain 对象，开发者需要在CF一端负责释放对象

**(\_\_bridge\_transfer T) op**：（ CF 转 ObjC 的时候使用）告诉编译器转移 CF 对象的所有权，开发者不再需要在CF一端负责释放对象

* 1. CoreAnimation

CoreAnimation使用CALayer来创建动画对象。动画在单独的线程中完成，不会阻塞主线程。动画只重绘界面上变化的部分（局部刷新）

* + 1. CALayer

所有的UIView都有一个默认的CALayer。[CALayer layer]

CALayer\* subLayer = [CALayer layer];

[view.layer addSublayer:subLayer];