Objective-C 和 Core Foundation 对象相互转换的内存管理总结

iOS允许Objective-C 和 Core Foundation 对象之间可以轻松的转换，拿 NSString 和 CFStringRef 来说，直接转换豪无压力：

```
• CFStringRef aCFString = (CFStringRef)aNSString;
• NSString *aNSString = (NSString *)aCFString;
```

针对内存管理问题，ARC 可以帮忙管理 Objective-C 对象，但是不支持 Core Foundation 对象的管理，所以转换后要注意一个问题：谁来释放使用后的对象。 本文重点总结一下类型转换后的内存管理。

一、非ARC的内存管理

倘若不使用ARC，手动管理内存，思路比较清晰，使用完，release转换后的对象即可。

```
1 //NSString 转 CFStringRef
2 CFStringRef aCFString = (CFStringRef) [[NSString alloc] initW
      ithFormat:@"%@", string];
```

```
3 //...
4 CFRelease(aCFString);
5
6
7 //CFStringRef 转 NSString
8 CFStringRef aCFString = CFStringCreateWithCString(kCFAllocato
      rDefault,
9                                                  bytes,
10                                                     NSUTF8S
      tringEncoding);
11    NSString *aNSString = (NSString *)aCFString;
12    //...
13    [aNSString release];
```

二、ARC下的内存管理

ARC的诞生大大简化了我们针对内存管理的开发工作，但是只支持管理 Objective-C 对象，不支持 Core Foundation 对象。Core Foundation 对象必须使用CFRetain和CFRelease来进行内存管理。那么当使用Objective-C 和 Core Foundation 对象相互转换的时候，必须让编译器知道，到底由谁来负责释放对象，是否交给ARC处理。只有正确的处理，才能避免内存泄漏和double free导致程序崩溃。

根据不同需求，有3种转换方式

1 __bridge （不改变对象所有权）

2 __bridge_retained 或者 CFBridgingRetain()（解除 ARC 所有权）

3 __bridge_transfer 或者 CFBridgingRelease()（给予 ARC 所有权）

1. __bridge_retained 或者 CFBridgingRetain()

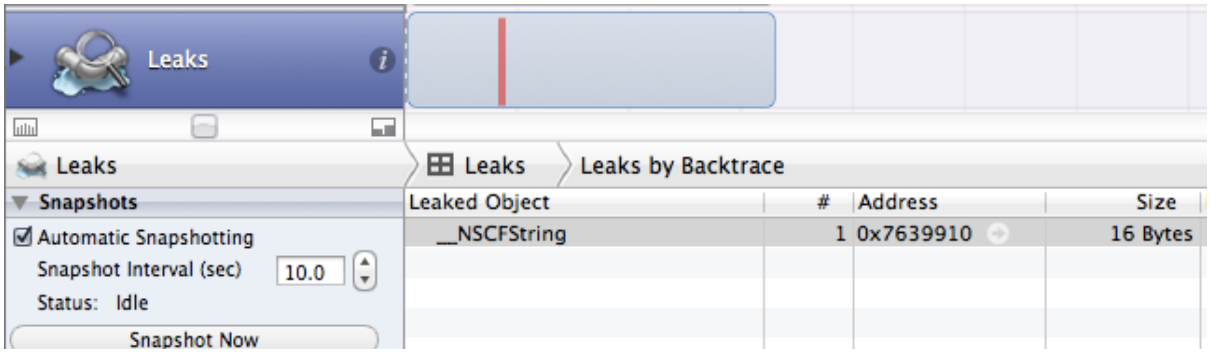
__bridge_retained 或者 CFBridgingRetain() 将Objective-C对象转换为Core Foundation对象，把对象所有权桥接给Core Foundation对象，同时剥夺ARC的管理权，后续需要开发者使用CFRelease或者相关方法手动来释放对象。

来看个例子：

```
1 - (void)viewDidLoad
2 {
3     [super viewDidLoad];
4
5     NSString *aNSString = [[NSString alloc]initWithFormat:@"t
      est"];
```

```
6     CFStringRef aCFString = (__bridge_retained CFStringRef) a
      NSString;
7
8     (void)aCFString;
9
10        //正确的做法应该执行CFRelease
11        //CFRelease(aCFString);
12    }
```

程序没有执行CFRelease，造成内存泄漏：



CFBridgingRetain() 是 __bridge_retained 的宏方法，下面两行代码等价：

```
1 CFStringRef aCFString = (__bridge_retained CFStringRef) aNSSt
      ring;
2 CFStringRef aCFString = (CFStringRef) CFBridgingRetain(aNSStr
      ing);
```

2. __bridge_transfer 或者 CFBridgingRelease()

__bridge_transfer 或者 CFBridgingRelease() 将非Objective-C对象转换为Objective-C对象，同时将对象的管理权交给ARC，开发者无需手动管理内存。

接着上面那个内存泄漏的例子，再转成OC对象交给ARC来管理内存，无需手动管理，也不会出现内存泄漏：

```
1 - (void)viewDidLoad
2 {
3     [super viewDidLoad];
4 
5     NSString *aNSString = [[NSString alloc]initWithFormat:@"t
      est"];
6     CFStringRef aCFString = (__bridge_retained CFStringRef) a
      NSString;
7     aNSString = (__bridge_transfer NSString *)aCFString;
8 }
```

1 CFBridgingRelease() 是__bridge_transfer的宏方法，下面两行代码等价：

2

```
1 aNSString = (__bridge_transfer NSString *)aCFString;
2 aNSString = (NSString *)CFBridgingRelease(aCFString);
```

3. __bridge

__bridge 只做类型转换，不改变对象所有权，是我们最常用的转换符。

从OC转CF，ARC管理内存：

```
1 - (void)viewDidLoad
2 {
3     [super viewDidLoad];
4 
5     NSString *aNSString = [[NSString alloc]initWithFormat:@"t
      est"];
6     CFStringRef aCFString = (__bridge CFStringRef)aNSString;

7 
8     (void)aCFString;
9 }
```

从CF转OC，需要开发者手动释放，不归ARC管：

```
1 - (void)viewDidLoad
2 {
3     [super viewDidLoad];
4 
5     CFStringRef aCFString = CFStringCreateWithCString(NULL, "
      test", kCFStringEncodingASCII);
6     NSString *aNSString = (__bridge NSString *)aCFString;
7 
8     (void)aNSString;
9 
10        CFRelease(aCFString);
11    }
```