设计模式 For iOS

第 02 式

单例模式

整理：BeyondVincent(破船)

时间：2013.05.09

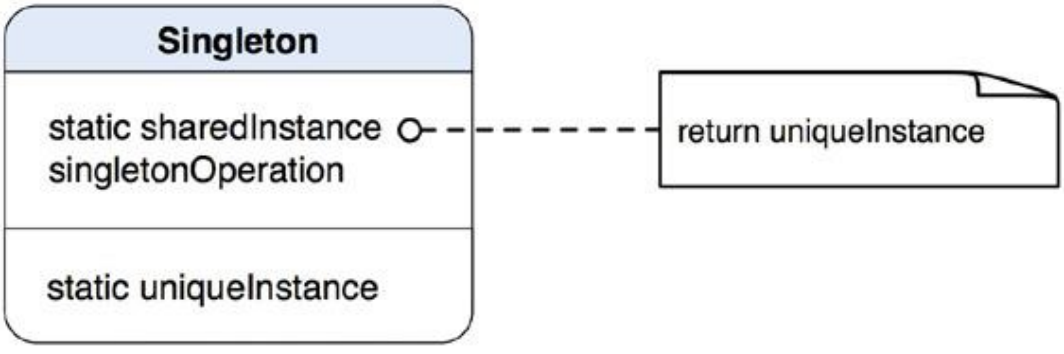
目录

第 02 式 单例模式

1.0. 简介

1.0.1. 什么是单例模式

单例模式是一个类在系统中只有一个实例对象。通过全局的一个入口点对这个实例对象进行访问。在 iOS 开发中，单例模式是非常有用的一种设计模式。如下图，是一个简单单例模式的 UML 类图。

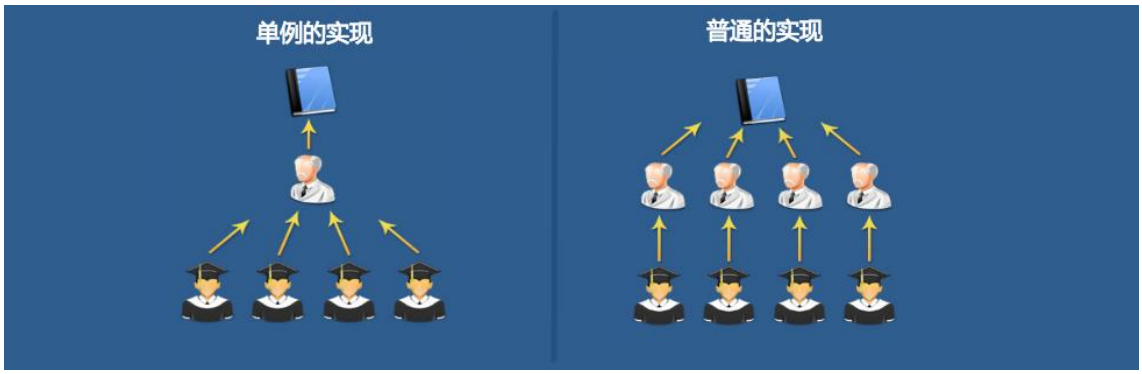


iOS SDK 中也有许多类使用了单例模式，例如，UIApplication：当程序启动的时候，会调用 UIApplicationMain 方法，在该方法中，会实例化一个 UIApplication 对象，之后在程序中的任意地方调用 sharedApplication 方法都将返回一个与当前应

用程序相关的 UIApplication 实例（UIApplicationMain 方法中创建的 UIApplication 单例）。

1.0.2. 什么时候使用单例模式？

在程序中，单例模式经常用于只希望一个类只有一个实例，而不运行一个类还有两个以上的实例。当然，在 iOS SDK 中，根据特定的需求，有些类不仅提供了单例访问的接口，还为开发者提供了实例化一个新的对象接口，例如，NSFileManager 可以通过 defaultManager 方法返回相同的一个 NSFileManager 对象。如果需要新的一个 NSFileManager 实例对象，可以通过 init 方法。



1.1. iOS 中单例模式的实现方法

iOS 中单例模式的实现方式一般分为两种：Non-ARC(非 ARC)和 ARC+GCD。

1)NON-ARC(非 ARC)

非 ARC 的实现方法如下所示：

BVNonARCSingleton.h

```
//
//  BVNonARCSingleton.h
//  SingletonPattern
//
//  Created by BeyondVincent on 13-5-9.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import &lt;Foundation/Foundation.h&gt;

@interface BVNonARCSingleton : NSObject

@property  ( nonatomic, retain) NSString  *tempProperty;
+ (BVNonARCSingleton *)sharedInstance;

@end
```

BVNonARCSingleton.m

```
//
//  BVNonARCSingleton.m
//  SingletonPattern
//
//  Created by BeyondVincent on 13-5-9.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVNonARCSingleton.h"

@implementation BVNonARCSingleton

static BVNonARCSingleton *sharedInstance = nil;

// 获取一个 sharedInstance 实例，如果有必要的话，实例化一个
+ (BVNonARCSingleton *)sharedInstance {
    if (sharedInstance == nil) {
        sharedInstance = [[super allocWithZone:NULL] init];
    }

    return sharedInstance;
}

// 当第一次使用这个单例时，会调用这个 init 方法。
- (id)init
{
    self = [super init];

    if (self) {
        // 通常在这里做一些相关的初始化任务
```

```
    }

    return self;
}

// 这个 dealloc 方法永远都不会被调用--因为在程序的生命周期内容，该单例一直都存在。(所以该方法可以不用实现)
-(void)dealloc
{
    [super dealloc];
}

// 通过返回当前的 sharedInstance 实例，就能防止实例化一个新的对象。
+ (id)allocWithZone:(NSZone*)zone {
    return [[self sharedInstance] retain];
}

// 同样，不希望生成单例的多个拷贝。
- (id)copyWithZone:(NSZone *)zone {
    return self;
}

// 什么也不做——该单例并不需要一个引用计数（retain counter）
- (id)retain {
    return self;
}

// 替换掉引用计数——这样就永远都不会 release 这个单例。
- (NSUInteger)retainCount {
    return NSUIntegerMax;
}

// 该方法是空的——不希望用户 release 掉这个对象。
- (oneway void)release {

}

//除了返回单例外，什么也不做。
- (id)autorelease {
    return self;
}

@end
```

实际上上面的代码苹果官网也有提供：Creating a Singleton Instance，只不过没有给出头文件的定义。上面用非 ARC 实现单例的方法是线程不安全的，如果有多个线程同时调用 sharedInstance 方法获取一个实例，而 sharedInstance 方法需要花费

1-2 秒钟的时间，那么 BVNonARCSingleton 的 init 方法就可能会被多次调用，也就是不同线程获得的 BVNonARCSingleton 有可能不是同一个实例。怎么解决线程的不安全呢？答案是使用@synchronized 来创建互斥锁即可。

```
// 保证在实例化的时候是线程安全的（当然，该方法不能保证该单例中所有方法的调用都是线程安全的）
@synchronized (self)
{
    if(sharedInstance == nil)
    {
        sharedInstance = [[super allocWithZone:NULL] init];
    }
}
```

通过上面的代码就能保存线程安全。

提醒：在 iOS 中，一般不建议使用非 ARC 来实现单例模式。更好的方法是使用 ARC+GCD 来实现。

2) ARC+GCD

通过 ARC+GCD 的方法来实现单例模式的非常简单的。下面先来看看具体实现：

BVARCSingleton.h

```
//
//  BVARCSingleton.h
//  SingletonPattern
//
//  Created by BeyondVincent on 13-5-9.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import &lt;Foundation/Foundation.h&gt;

@interface BVARCSingleton : NSObject

@property  ( nonatomic, weak) NSString  *tempProperty;
```

```
+ (BVARCSingleton *)sharedInstance;

@end
```

BVARCSingleton.m

```
//
//  BVARCSingleton.m
//  SingletonPattern
//
//  Created by BeyondVincent on 13-5-9.
//  Copyright (c) 2013 年 BeyondVincent. All rights reserved.
//

#import "BVARCSingleton.h"

@implementation BVARCSingleton

+ (BVARCSingleton *) sharedInstance
{
    static  BVARCSingleton *sharedInstance = nil ;
    static  dispatch_once_t onceToken;  // 锁
    dispatch_once (&amp;amp; onceToken, ^ {      // 最多调用一次
        sharedInstance = [[self  alloc] init];
    });
    return  sharedInstance;
}

// 当第一次使用这个单例时，会调用这个 init 方法。
- (id)init
{
    self = [super init];

    if (self) {
        // 通常在这里做一些相关的初始化任务
    }

    return self;
}

@end
```

在上面的代码中，调用 Grand Central Dispatch (GCD)中的 dispatch_once 方法就可以确保 BVARCSingleton 只被实例化一次。并且该方法是线程安全的，我们不

用担心在不同的线程中，会获得不同的实例。（当然，该方法同样不能保证该单例中所有方法的调用都是线程安全的）。

当然，在 ARC 中，不用 GCD 也是可以做到线程安全的，跟之前非 ARC 代码中使用@synchronized 一样，如下代码：

```
// 不使用 GCD，通过@synchronized
@synchronized (self)
{
    if(sharedInstance == nil)
    {
        sharedInstance = [[self alloc] init];
    }
}
```

为了简化使用 ARC+GCD 来创建单例，可以定义下面这样的一个宏：

```
#define DEFINE_SHARED_INSTANCE_USING_BLOCK(block) \
static dispatch_once_t onceToken = 0; \
__strong static id sharedInstance = nil; \
dispatch_once(&amp;onceToken, ^{ \
sharedInstance = block(); \
}); \
return sharedInstance; \
```

实例化的实现方法如下所示：

```
+ (BVARCSingleton *) sharedInstance
{
    DEFINE_SHARED_INSTANCE_USING_BLOCK(^{
        return [[self alloc] init];
    });
}
```

1.2. 单例的使用

单例的使用方法很简单，在代码中的任意位置，如下使用即可：

在 BVAppDelegate.m 中添加头文件：

```
#import "BVNonARCSingleton.h"
#import "BVARCSingleton.h"
```

如下使用方法：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    [BVNonARCSingleton sharedInstance].tempProperty = @"非 ARC 单例的实现";
    NSLog(@"%@", [BVNonARCSingleton sharedInstance].tempProperty);

    [BVARCSingleton sharedInstance].tempProperty = @"ARC 单例的实现";
    NSLog(@"%@", [BVARCSingleton sharedInstance].tempProperty);

    return YES;
}
```

运行程序，会在控制台窗口输出如下内容：

```
2013-05-09 16:44:07.649 SingletonPattern[5159:c07]  非 ARC 单例的实现
2013-05-09 16:44:33.204 SingletonPattern[5159:c07] ARC 单例的实现
```

1.3. 代码下载地址

点击如下图标，浏览并下载本文全部代码。

1.4. 参考

本文参考了如下文章：

1. 本文在写作的时候，参考了许多网上的优秀文章，包括如下：
2. What Is the Singleton Pattern?（Pro Objective-C Design Patterns for iOS 书中对单例模式的介绍
3. ios-patterns-singleton（需翻墙，是俄文的）
4. Singletons in Objective-C
5. Implementing a Singleton in Objective-C / iOS
6. Grand Central Dispatch (GCD) Reference（dispatch_once 的介绍）
7. 使用 GCD（来自唐巧的一篇文章，对 GCD 总结不错）
8. Threading Programming Guide（@synchronized 关键字的介绍）
9. Cocoa Fundamentals Guide（苹果官方给的一个非 ARC 单例实现）

关于设计模式 For iOS 的整理

本系列文章，主要是学习设计模式在 iOS 中的实现过程中，写出来的。

期间参考了许多互联网上的资料。如有不正确的地方，还请读者指正。

本系列全部文章和相关代码都可以在下面的链接中下载到：

https://github.com/BeyondVincent/ios_patterns

感谢你的阅读！

如果对这篇文章有问题和建议，可以与我联系：

你可以发邮件与破船取得联系: BeyondVincent@gmail.com
还可以关注破船的微博: 腾讯微博和新浪微博。
这里是破船的个人博客，欢迎光临：破船之家