1、写出方法获取ios内存使用情况。  
// 获取当前设备可用内存及所占内存的头文件

#import <sys/sysctl.h>

#import <mach/mach.h>

// 获取当前设备可用内存(单位：MB）

- (double)availableMemory

{

  vm\_statistics\_data\_t vmStats;

  mach\_msg\_type\_number\_t infoCount = HOST\_VM\_INFO\_COUNT;

  kern\_return\_t kernReturn = host\_statistics(mach\_host\_self(),

                                             HOST\_VM\_INFO,

                                             (host\_info\_t)&vmStats,

                                             &infoCount);

  if (kernReturn != KERN\_SUCCESS) {

    return NSNotFound;

  }

  return ((vm\_page\_size \*vmStats.free\_count) / 1024.0) / 1024.0;

}

// 获取当前任务所占用的内存（单位：MB）

- (double)usedMemory

{

  task\_basic\_info\_data\_t taskInfo;

  mach\_msg\_type\_number\_t infoCount = TASK\_BASIC\_INFO\_COUNT;

  kern\_return\_t kernReturn = task\_info(mach\_task\_self(),

                                       TASK\_BASIC\_INFO,

                                       (task\_info\_t)&taskInfo,

                                       &infoCount);

  if (kernReturn != KERN\_SUCCESS

      ) {

    return NSNotFound;

  }

  return taskInfo.resident\_size / 1024.0 / 1024.0;}

1. 深拷贝和浅拷贝的理解？  
   深拷贝拷贝的是内容，浅拷贝拷贝的是指针。深拷贝和浅拷贝最大的区别就是子类对象的地址是否改变，如果子类对象的地址改变那么就是深拷贝。

2、怎样实现一个singleton的类。  
static LOSingleton \* shareInstance;

+( LOSingleton \*)sharedInstance{

@synchronized(self){//这个东西其实就是 一个加锁。如果self 其他线程访问，则会阻塞。这样做一般是用来对单例 进行一个死锁的保护

if (shareInstance == nil) {

shareInstance = [[super allocWithZone:NULL] init];

}

}

return shareInstance;

}

//第二种方式

+ (LOSingleton \*) sharedInstance

{

static LOSingleton \*sharedInstance = nil ;

static dispatch\_once\_t onceToken; // 锁

dispatch\_once (& onceToken, ^ { // 最多调用一次

sharedInstance = [[self alloc] init];

});

return sharedInstance;

}

3、什么是安全释放？

在对象release之后把指针置为nil

1. RunLoop是什么？  
   一个RunLoop就是一个事件处理的循环，用来不停的调度工作以及处理输入时间。使用runloop的目的是让你的线程在有工作的时候忙于工作,而没工作的时候处于休眠状态。runloop的设计是为了减少cpu无谓的空转。
2. 什么是序列化和反序列化，可以用来做什么？如何在OC中实现复杂对象的存储？  
   如果你需要存储一个复杂的对象的话，经常要以二进制的方法序列化这个对象，这个过程叫Archiving。如果一个对象需要进行序列化，那么需要遵循NScoding协议，主要有两个方法：  
   -(id)initWithCoder:(NSCoder\*)coder;//从coder中读取数据，保存到相应变量中，即反序列化数据。  
   -(void)encodeWithCoder:(NSCoder\*)coder;//读取实例变量，并把这些数据写到coder中去，即序列化数据。
3. 写一个标准宏MIN，这个宏输入两个参数并返回较小的一个？  
   #define kMIN(X,Y) ((X) > (Y)) ? (Y) :(X)
4. iphone os有没有垃圾回收机制？简单阐述一下OC内存管理。  
   iphone os没有垃圾回收机制。  
   垃圾回收机制用于在空闲时间以不定时的方式动态的回收无任何引用的对象占据的内存空间。
5. 简述应用程序按Home键进入后台时的生命周期，以及从后台回到前台时的生命周期？  
   应用程序的状态：  
   Not running 未运行，程序没启动  
   Inactive 未激活，程序在前台运行，不过没接受到事件，没有事件处理的状态下通常处于这个状态。  
   Active 激活 程序在前台并且接收到了事件  
   Backgound 后台 程序在后台而且能执行代码，大多数程序进入这个状态后会在在这个状态上停留一会。  
   Suspended 挂起 程序在后台不能执行代码。  
   - (BOOL)application:(UIApplication \*)application willFinishLaunchingWithOptions:(NSDictionary \*)launchOptions     //告诉代理进程启动但还没进入状态保存 - (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions   // 告诉代理启动基本完成程序准备开始运行 - (void)applicationWillResignActive:(UIApplication \*)application   // 当应用程序将要入非活动状态执行，在此期间，应用程序不接收消息或事件，比如来电话了 - (void)applicationDidBecomeActive:(UIApplication \*)application      //  当应用程序入活动状态执行，这个刚好跟上面那个方法相反 - (void)applicationDidEnterBackground:(UIApplication \*)application   //  当程序被推送到后台的时候调用。所以要设置后台继续运行，则在这个函数里面设置即可 - (void)applicationWillEnterForeground:(UIApplication \*)application  //当程序从后台将要重新回到前台时候调用，这个刚好跟上面的那个方法相反。 - (void)applicationWillTerminate:(UIApplication \*)application  //当程序将要退出是被调用，通常是用来保存数据和一些退出前的清理工作。这个需要要设置UIApplicationExitsOnSuspend的键值。 - (void)applicationDidFinishLaunching:(UIApplication\*)application  //当程序载入后执行
6. ViewController 的 alloc，loadView, viewDidLoad,viewWillAppear,viewDidUnload,dealloc、init分别是在什么时候调用的？在自定义ViewController的时候这几个函数里面应该做什么工作？

alloc初始化当前的ViewController

loadView:没有正在使用nib视图页面，子类将会创建自己的自定义视图层

viewDidLoad:试图被加载后调用

viewWillAppear：试图即将出现的时候调用

viewDidUnload：<iOS6之后废弃>当系统内存吃紧的时候会调用该方法,释放掉当前未在window中显示的试图和对应的控制器