1. **Difference between shallow copy and deep copy?**
   浅复制和深复制的区别？

答：浅复制是指复制对象的指针，不复制对象本身。深复制是指复制引用对象本身。

意思就是说我有个 A 对象，复制一份后得到 A_copy 对象后，对于浅复制来说，A 和 A_copy 指向的是同一个内存资源，复制的只不过是一个指针，对象本身资源还是只有一份，那如果我们对 A_copy 执行了修改操作,那么发现 A 引用的对象同样被修改，这其实违背了我们复制拷贝的一个思想。深复制就好理解了,内存中存在了两份独立对象本身。

2. **What is advantage of categories? What is difference between implementing a category and inheritance?**
   类别的作用？继承和类别在实现中有何区别？
   答：categories 是可以在不知道，不改变原来代码的情况下，往里面添加新的方法。只能添加，不能删除和修改。并且，如果类别和原来类中的方法名冲突，则类别将覆盖原方法，因为类别具有更高的优先级。
   类别主要有 3 个作用：(1)：将类的实现分散到多个不同文件或者多个不同框架中。
   (2)：创建对私有方法的前向引用。
   (3)：向对象添加非正式协议。
   继承可以添加，修改方法，并且可以增加属性。
3. **Difference between categories and extensions?**
   类别和类扩展（延展）的区别？
   答：相同点：都可以添加属性和方法。
   不同点：延展可以合成属性，可以添加实例变量。
4. **Object－c 的类可以多重继承么？可以实现多个接口么？Category 是什么？重写一个类的方式用继承好还是分类好？为什么？**
   答：OC 中的类不可以多重继承，可以用 protocol 委托代理来完成多继承。可以实现多个接口。Category 是类别。一般情况下重写一个方法用分类比较好，用 Category 重写类的方法，仅对本 category 有效，不会影响到其他类与原有类的关系。

5.#import 跟#include 又什么区别，@class 呢， ＃import<> 跟 #import”″ 又什么区别？

答：#import 是 Object——C 导入头文件的关键字，#include 是 C 和 C++导入头文件的关键字。使用 import 头文件会自动只导入一次，不会重复导入，不会产生重复编译。

@class 告诉编译器某个类的声明，当执行时，才去查看类的实现文件，可以解决头文件的相互包含。

＃import<>用来引入类库或者框架的头文件，#import“ “用来引入非类库的头文件。

6. 属性 readwrite，readonly，assign，retain，copy，nonatomic 各是什么作用，在那种情况下用？

答：readwrite：是可读可写特性，在需要生成 getter 和 setter 方法时用。

readonly：是只读特性，只会生成 getter 方法，不会生成 setter 方法，不希望属性在类外改变。

Assign：是赋值特性，setter 方法将传入参数赋值给实例变量。仅在设置变量时用。

Retain：表示持有特性，setter 方法将传入参数先保留再赋值，传入参数的 retaincount 会加 1。

Copy：表示赋值特性，setter 方法将传入对象复制一份。在需要完全一份新的变量时候使用。

Nonatomic：非原子操作，决定编译器生成的 getter 和 setter 是否是原子操作。

Atomic：表示多线程安全，一般使用 nonatomic。

7.内存管理的几条原则时什么？按照默认法则.那些关键字生成的对象需要手动释放？在和 property 结合的时候怎样有效的避免内存泄露？

答：原则：1 如果使用 alloc 或者 copy,new 方法创建的对象，或者使用 retain 保留一个对象，那么都要自己释放对象。（即：谁申请，谁释放）

2：在大多数情况下，申请内存的语句数量和释放内存的语句数量应该相等。

3：尽量少使用内存，用完后立即释放。

关键字：create，copy，alloc 和 new 生成的对象需要手动释放。

设置正确的 property 属性，对于 retain 需要在合适的地方释放，可以有效避免内存泄漏。

8.What is purpose of delegates?代理的作用？

答：代理的目的是改变或传递控制链。允许一个类在某些特定时刻通知到其他类，而不需要获取到那些类的指针。可以减少框架的复杂度。

9：static 关键字的作用？

答：(1)：在函数体内，static 变量的作用局限为该函数体，该变量的内存只被分配一次。因此其值在下次调用时仍维持上次的值。

(2)：在模块内，static 全局变量可以被模块内所用函数访问，但不能被模块外的其他函数访问。

(3)：在模块内的 static 函数只可被这一模块内的其他函数调用，这个函数的使用局限被限制在声明他的模块内。

(4)：在类中的 static 成员变量属于整个类所拥有，对类的所有对象唯有一份 copy。

(5)：在类中的 static 成员函数属于整个类所拥有，这个函数不接收 this 指针，因而只能访问类的 static 成员变量。

10：关键字 const 什么含义

const 意味着”只读”，

下面的声明都是什么意思？

```
const int a;
int const a;
const int *a;
int * const a;
int const * a const;
```

前两个的作用是一样，a 是一个常整型数。第三个意味着 a 是一个指向常整型数的指针(也就是，整型数是不可修改的，但指针可以)。第四个意思 a 是一个指向整型数的常指针（也就是说，指针指向的整型数是可以修改的，但指针是不可修改的）。最后一个意味着 a

是一个指向常整型数的常指针（也就是说，指针指向的整型数是不可修改的，同时指针也是不可修改的）。

结论：

●; 关键字 const 的作用是为给读你代码的人传达非常有用的信息，实际上，声明一个参数为常量是为了告诉了用户这个参数的应用目的。如果
你曾花很多时间清理其它人留下的垃圾，你就会很快学会感谢这点多余的信息。（当然，懂得用 const 的程序员很少会留下的垃圾让别人来清
理的。）

●; 通过给优化器一些附加的信息，使用关键字 const 也许能产生更紧凑的代码。

●; 合理地使用关键字 const 可以使编译器很自然地保护那些不希望被改变的参数，防止其被无意的代码修改。简而言之，这样可以减少 bug 的出现。

欲阻止一个变量被改变，可以使用 const 关键字。在定义该 const 变量时，通常需要对它进行初
始化，因为以后就没有机会再去改变它了；

（2）对指针来说，可以指定指针本身为 const，也可以指定指针所指的数据为 const，或二者同时指
定为 const；

（3）在一个函数声明中，const 可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；

（4）对于类的成员函数，若指定其为 const 类型，则表明其是一个常函数，不能修改类的成员变量；

（5）对于类的成员函数，有时候必须指定其返回值为 const 类型，以使得其返回值不为"左值"。

11：关键字 volatile 有什么含义？并给出三个不同例子？

答：一个定义为 volatile 的变量是说这变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值了。精确地说就是，优化器在用到
这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。下面是 volatile 变量的几个例子：

- 并行设备的硬件寄存器（如：状态寄存器）
- 一个中断服务子程序中会访问到的非自动变量(Non-automatic

variables)

- 多线程应用中被几个任务共享的变量

12● 一个参数既可以是 const 还可以是 volatile 吗？解释为什么。

答：是的。一个例子是只读的状态寄存器。它是 **volatile** 因为它可能被意想不到地改变。它是 **const** 因为程序不应该试图去修改它。

13● 一个指针可以是 volatile 吗？解释为什么。

答： 是的。尽管这并不很常见。一个例子是当一个中服务子程序修该一个指向一个 **buffer** 的指针时。

14：线程和进程的区别？

进程和线程都是由操作系统所体会的程序运行的基本单元，系统利用该基本单元实现系统对应用的并发性。

进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。

15：堆和栈的区别？

管理方式：对于栈来讲，是由编译器自动管理，无需我们手工控制；对于堆来说，释放工作由程序员控制，容易产生 **memory leak**。

申请大小：

栈：在 **Windows** 下,栈是向低地址扩展的数据结构，是一块连续的内

存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在 WINDOWS 下，栈的大小是 2M（也有的说是 1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示 overflow。因此，能从栈获得的空间较小。

堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

碎片问题：对于堆来讲，频繁的 new/delete 势必会造成内存空间的不连续，从而造成大量的碎片，使程序效率降低。对于栈来讲，则不会存在这个问题，因为栈是先进后出的队列，他们是如此的一一对应，以至于永远都不可能有一个内存块从栈中间弹出

分配方式：堆都是动态分配的，没有静态分配的堆。栈有 2 种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配。动态分配由 alloca 函数进行分配，但是栈的动态分配和堆是不同的，他的动态分配是由编译器进行释放，无需我们手工实现。

分配效率：栈是机器系统提供的数据结构，计算机会在底层对栈提供支持：分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，这就决定了栈的效率比较高。堆则是 C/C++函数库提供的，它的机制是很复杂的。

16：Object-C 的内存管理？

1.当你使用 new,alloc 和 copy 方法创建一个对象时,该对象的保留计数器值为 1.当你不再使用该对象时,你要负责向该对象发送一条 release 或 autorelease 消息.这样,该对象将在使用寿命结束时被销毁.

2.当你通过任何其他方法获得一个对象时,则假设该对象的保留计数器值为 1,而且已经被设置为自动释放,你不需要执行任何操作来确保

该对象被清理.如果你打算在一段时间内拥有该对象,则需要保留它并确保在操作完成时释放它.

3.如果你保留了某个对象,你需要(最终)释放或自动释放该对象.必须保持 retain 方法和 release 方法的使用次数相等.

17：对象是什么时候被 dealloc 的？

引用计数为 0 时。

autorelease 实际上只是把对 release 的调用延迟了，对于每一个 Autorelease，系统只是把该 Object 放入了当前的 Autorelease pool 中，当该 pool 被释放时，该 pool 中的所有 Object 会被调用 Release。对于每一个 Runloop，系统会隐式创建一个 Autorelease pool，这样所有的 release pool 会构成一个象 CallStack 一样的一个栈式结构，在每一个 Runloop 结束时，当前栈顶的 Autorelease pool 会被销毁，这样这个 pool 里的每个 Object（就是 autorelease 的对象）会被 release。那什么是一个 Runloop 呢？ 一个 UI 事件，Timer call， delegate call， 都会是一个新的 Runloop

18：iOS 有没有垃圾回收？

Objective-C 2.0 也是有垃圾回收机制的，但是只能在 Mac OS X Leopard 10.5 以上的版本使用。

19：怎么理解 MVC，在 Cocoa 中 MVC 是怎么实现的？

MVC 设计模式考虑三种对象：模型对象、视图对象、和控制器对象。模型对象代表特别的知识和专业技能，它们负责保有应用程序的数据和定义操作数据的逻辑。视图对象知道如何显示应用程序的模型数据，而且可能允许用户对其进行编辑。控制器对象是应用程序的视图对象和模型对象之间的协调者。

20：id、nil 代表什么？

id 和 void *并非完全一样。在上面的代码中，id 是指向 struct objc_object 的一个指针，这个意思基本上是说，id 是一个指向任何一个继承了 Object（或者 NSObject）类的对象。需要注意的是 id 是一个指针，所以你在使用 id 的时候不需要加星号。比如 id foo=nil 定义了一个 nil 指针，这个指针指向 NSObject 的一个任意子类。而 id *foo=nil 则定义了一个指针，这个指针指向另一个指针，被指向的这个指针指向 NSObject 的一个子类。

nil 和 C 语言的 NULL 相同，在 objc/objc.h 中定义。nil 表示一个 Objctive-C 对象，这个对象的指针指向空（没有东西就是空）。

21：Object-C 有私有方法吗？私有变量呢？

objective-c – 类里面的方法只有两种，静态方法和实例方法．这似乎就不是完整的面向对象了,按照 OO 的原则就是一个对象只暴露有用的东西．如果没有了私有方法的话，对于一些小范围的代码重用就不那么顺手了．在类里面声名一个私有方法

```
@interface Controller : NSObject { NSString *something; }
+ (void)thisIsAStaticMethod;
– (void)thisIsAnInstanceMethod;
@end
@interface Controller (private) -
(void)thisIsAPrivateMethod;
@end
```

@private 可以用来修饰私有变量

在 Objective - C 中，所有实例变量默认都是私有的，所有实例方法默认都是公有的

22：retaincount？关于引用计数？

答：1：当通过 alloc 和 new 初始化一个对象的时候，引用计数为 1.

2：当对该对象 retain 或者被另外一个对象保留或持有的时候（数组），引用计数加 1.

3;当用 release 或 autorelease 或其他对象取消持有时，引用计数减 1.

4：当引用计数为 0 时，系统会自动调用 dealloc 方法来释放该对象。

23：self.跟 self 有什么区别？

24：什么是 KVO 和 KVC?

答案：kvc:键 - 值编码是一种间接访问对象的属性使用字符串来标识属性，而不是通过调用存取方法，直接或通过实例变量访问的机制。 很多情况下可以简化程序代码。 kvo:键值观察机制，他提供了观察某一属性变化的方法，极大的简化了代码。

25：自动释放池是什么,如何工作？

当您向一个对象发送一个autorelease消息时，Cocoa就会将该对 象的一个引用放入到最新的自动释放池。它仍然是个正当的对象，因此自动释放池定义的作用域内的其它对象可以向它发送消息。当程序执行到作用域结束的位置 时，自动释放池就会被释放，池中的所有对象也就被释放。

1. ojc-c 是通过一种"referring counting"(引用计数)的方式来管理内存的，对象在开始分配内存(alloc)的时候引用计数为一,以后每当碰到有copy,retain的时候引用计数都会加一，每当碰到release和autorelease的时候引用计数就会减一,如果此对象的计数变为了0，就会被系统销毁. 2. NSAutoreleasePool 就是用来做引用计数的管理工作的,这个东西一般不用你管的. 3. autorelease和release没什么区别,只是引用计数减一的时机不同而已,autorelease会在对象的使用真正结束的时候才做引用计数 减一.

26：What are mutable and immutable types in Objective C? obc 中可修改和不可以修改类型。

答案：可修改不可修改的集合类。这个我个人简单理解就是可动态添加修改和不可动态添加修改一样。 比如 NSArray 和 NSMutableArray。前者在初始化后的内存控件就是固定不可变的，后者可以添加等，可以动态申请新的内存空间。

27.Difference between frame and bounds? frame 和 bounds 有什么不同？

答案:frame 指的是：该 view 在父 view 坐标系统中的位置和大小。（参照点是父亲的坐标系统） bounds 指的是：该 view 在本身坐标系统中的位置和大小。（参照点是本身坐标系统）

28.Difference between method and selector? 方法和选择器有何不同？

答案：selector是一个方法的名字，method是一个组合体，包含了名字和实现.

29.What is lazy loading?

答案：懒汉模式，只在用到的时候才去初始化。 也可以理解成延时加载。 我觉得最好也最简单的一个列子就是 tableView 中图片的加载显示了。 一个延时载，避免内存过高，一个异步加载，避免线程堵塞。

30：ios 平台怎么做数据的持久化?coredata 和 sqlite 有无必然联系？coredata 是一个关系型数据库吗？

iOS 中可以有四种持久化数据的方式：属性列表、对象归档、SQLite3 和 Core Data；core data 可以使你以图形界面的方式快速的定义 app 的数据模型，同时在你的代码中容易获取到它。core data 提供了基础结构去处理常用的功能，例如保存，恢复，撤销和重做，允许你在 app 中继续创建新的任务。在使用 core data 的时候，你不用安装额外的数据库系统，因为 core data 使用内置的 sqlite 数据库。core data 将你 app 的模型层放入到一组定义在内存中的数据对象。core data 会追踪这些对象的改变，同时可以根据需要做相反的改变，例如用户

执行撤销命令。当 core data 在对你 app 数据的改变进行保存的时候，core data 会把这些数据归档，并永久性保存。
mac os x 中 sqlite 库，它是一个轻量级功能强大的关系数据引擎，也很容易嵌入到应用程序。可以在多个平台使用，sqlite 是一个轻量级的嵌入式 sql 数据库编程。与 core data 框架不同的是，sqlite 是使用程序式的，sql 的主要的 API 来直接操作数据表。
Core Data 不是一个关系型数据库，也不是关系型数据库管理系统(RDBMS)。虽然 Core Dta 支持 SQLite 作为一种存储类型，但它不能使用任意的 SQLite 数据库。Core Data 在使用的过程种自己创建这个数据库。Core Data 支持对一、对多的关系。

31：What is responder chain? 说说响应链

答案：事件响应链。包括点击事件，画面刷新事件等。在视图栈内从上至下，或者从下之上传播。 可以说点事件的分发，传递以及处理。

32：what is difference between NSNotification and protocol? 通知和协议的不同之处？

答案：协议有控制链(has-a)的关系，通知没有。 首先我一开始也不太明白，什么叫控制链（专业术语了~）。但是简单分析下通知和代理的行为模式，我们大致可以有自己的理解 简单来说，通知的话，它可以一对多，一条消息可以发送给多个消息接受者。 代理按我们的理解，到不是直接说不能一对多，比如我们知道的明星经济代理人，很多时候一个经济人负责好几个明星的事务。 只是对于不同明星间，代理的事物对象都是不一样的，一一对应，不可能说明天要处理 A 明星要一个发布会，代理人发出处理发布会的消息后，别称 B 的 发布会了。但是通知就不一样，他只关心发出通知，而不关心多少接收到感兴趣要处理。因此控制链（has-a 从英语单词大致可以看出，单一拥有和可控制的对应关系。

33. id 声明的对象有什么特性？

答：Id 声明的对象具有运行时的特性，即可以指向任意类型的 objcetive-c 的对象；

34. Objective-C 如何对内存管理的，说说你的看法和解决方法？

答：Objective-C 的内存管理主要有三种方式 ARC（自动内存计数）、手动内存计数、内存池。

35.what is difference between NSNotification and protocol?

通知和协议的不同之处？

答案：协议有控制链(has-a)的关系，通知没有。

首先我一开始也不太明白，什么叫控制链（专业术语了~）。但是简单分析下通知和代理的行为模式，我们大致可以有自己的理解

简单来说，通知的话，它可以一对多，一条消息可以发送给多个消息接受者。

代理按我们的理解，到不是直接说不能一对多，比如我们知道的明星经济代理人，很多时候一个经济人负责好几个明星的事务。

只是对于不同明星间，代理的事物对象都是不一样的，对应，不可能说明天要处理 A 明星要一个发布会，代理人发出处理发布会的消息后，别称 B 的

发布会了。但是通知就不一样，他只关心发出通知，而不关心多少接收到感兴趣要处理。

因此控制链（has-a 从英语单词大致可以看出，单一拥有和可控制的对应关系。

36.What is push notification?什么是推送消息？

答：推送通知更是一种技术。简单点就是客户端获取资源的一种手段。普通情况下，都是客户端主动的 pull。推送则是服务器端主动 push。

37.Polymorphism?关于多态性

38.Singleton?对于单例的理解

39.Can we use two tableview controllers on one viewcontroller?

是否在一个视图控制器中嵌入两个 tableview 控制器？

答案：一个视图控制只提供了一个 View 视图，理论上一个

tableViewController 也不能放吧，

只能说可以嵌入一个 tableview 视图。当然，题目本身也有歧义，如

果不是我们定性思维认为的 UIViewController，

而是宏观的表示视图控制者，那我们倒是可以把其看成一个视图控制

者，它可以控制多个视图控制器，比如 TabbarController 那样的感觉

40.Can we use one tableview with two different datasources? How you will achieve this?

一个 tableView 是否可以关联两个不同的数据源？你会怎么处理？

答案：首先我们从代码来看，数据源如何关联上的，其实是在数据源关联的代理方法里实现的。

因此我们并不关心如何去关联他，他怎么关联上，方法只是让我返回根据自己的需要去设置如相关的数据源。

因此，我觉得可以设置多个数据源啊，但是有个问题是，你这是想干嘛呢？想让列表如何显示，不同的数据源分区块显示？

41:tableView 的重用机制？

查看 UITableView 头文件，会找到 NSMutableArray* visiableCells，和 NSMutableDictnery* reusableTableCells 两个结构。visiableCells 内保存当前显示的 cells，reusableTableCells 保存可重用的 cells。

TableView 显示之初，reusableTableCells 为空，那么 tableView dequeueReusableCellWithIdentifier:CellIdentifier 返回 nil。开始的 cell 都是通过 [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier]来创建，而且 cellForRowAtIndexPath 只是调用最大显示 cell 数的次数。

比如：有 100 条数据，iPhone 一屏最多显示 10 个 cell。程序最开始显示 TableView 的情况是：

1,用[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier]创建 10 次 cell，并给 cell 指定同样的重用标识(当然，可以为不同显示类型的 cell 指定不同的标识)。并且 10 个 cell 全部都加入到 visiableCells 数组，reusableTableCells 为空。

2,向下拖动 tableView，当 cell1 完全移出屏幕，并且 cell11(它也是 alloc 出来的，原因同上)完全显示出来的时候。cell11 加入到 visiableCells，cell1 移出 visiableCells，cell1 加入到 reusableTableCells。

3. 接着向下拖动 tableView，因为 reusableTableCells 中已经有值，所以，当需要显示新的 cell，cellForRowAtIndexPath 再次被调用的时候，tableView dequeueReusableCellWithIdentifier:CellIdentifier，返回 cell1。cell1 加入到 visiableCells，cell1 移出 reusableTableCells；cell2 移出 visiableCells，cell2 加入到 reusableTableCells。之后再需要显示的 Cell 就可以正常重用了。

42:ViewController 的 loadView、viewDidLoad、viewDidUnload 分别是什么时候调用的，在自定义 ViewCointroller 时在这几个函数中应该做什么工作？

由 init、loadView、viewDidLoad、viewDidUnload、dealloc 的关系说起
init 方法
在 init 方法中实例化必要的对象（遵从 LazyLoad 思想）
init 方法中初始化 ViewController 本身
loadView 方法
当 view 需要被展示而它却是 nil 时，viewController 会调用该方法。不要直接调用该方法。
如果手工维护 views，必须重载重写该方法
如果使用 IB 维护 views，必须不能重载重写该方法
loadView 和 IB 构建 view
你在控制器中实现了 loadView 方法，那么你可能会在应用运行的某个时候被内存管理控制调用。 如果设备内存不足的时候， view 控制器会收到 didReceiveMemoryWarning 的消息。 默认的实现是检查当前控制器的 view 是否在使用。 如果它的 view 不在当前正在使用的 view hierarchy 里面，且你的控制器实现了 loadView 方法，那么这个 view 将被 release, loadView 方法将被再次调用来创建一个新的 view。
viewDidLoad 方法
viewDidLoad 此方法只有当 view 从 nib 文件初始化的时候才被调用。
重载重写该方法以进一步定制 view
在 iPhone OS 3.0 及之后的版本中，还应该重载重写 viewDidUnload 来释放对 view 的任何索引
viewDidLoad 后调用数据 Model
viewDidUnload 方法
当系统内存吃紧的时候会调用该方法（注：viewController 没有被 dealloc）
内存吃紧时，在 iPhone OS 3.0 之前 didReceiveMemoryWarning 是释放无用内存的唯一方式，但是 OS 3.0 及以后 viewDidUnload 方法是更好的方式
在该方法中将所有 IBOutlet（无论是 property 还是实例变量）置为 nil（系统 release view 时已经将其 release 掉了）
在该方法中释放其他与 view 有关的对象、其他在运行时创建（但非系统必须）的对象、在 viewDidLoad 中被创建的对象、缓存数据等 release 对象后，将对象置为 nil（IBOutlet 只需要将其置为 nil，系统 release view 时已经将其 release 掉了）
一般认为 viewDidUnload 是 viewDidLoad 的镜像，因为当 view 被重新请求时，viewDidLoad 还会重新被执行
viewDidUnload 中被 release 的对象必须是很容易被重新创建的对象（比如在 viewDidLoad 或其他方法中创建的对象），不要 release 用户数据或其他很难被重新创建的对象

dealloc 方法
viewDidUnload 和 dealloc 方法没有关联，dealloc 还是继续做它该做的事情

43:ViewController 的 didReceiveMemoryWarning 是在什么时候调用的？默认的操作是什么？

当程序接到内存警告时 View Controller 将会收到这个消息：
didReceiveMemoryWarning
从 iOS3.0 开始，不需要重载这个函数，把释放内存的代码放到 viewDidUnload 中去。
这个函数的默认实现是:检查 controller 是否可以安全地释放它的 view(这里加粗的 view 指的是 controller 的 view 属性)，比如 view 本身没有 superview 并且可以被很容易地重建（从 nib 或者 loadView 函数）。
如果 view 可以被释放，那么这个函数释放 view 并调用 viewDidUnload。
你可以重载这个函数来释放 controller 中使用的其他内存。但要记得调用这个函数的 super 实现来允许父类（一般是 UIVIewController）释放 view。
如果你的 ViewController 保存着 view 的子 view 的引用，那么，在早期的 iOS 版本中，你应该在这个函数中来释放这些引用。而在 iOS3.0 或更高版本中，你应该在 viewDidUnload 中释放这些引用。

44:KVC 和 KVO 区别，分别在什么情况下使用？

KVC(Key-Value-Coding)
KVO（Key-Value-Observing）
理解 KVC 与 KVO（键-值-编码与键-值-监看）
当通过 KVC 调用对象时，比如：[self valueForKey:@"someKey"]时，程序会自动试图通过几种不同的方式解析这个调用。首先查找对象是否带有 someKey 这个方法，如果没找到，会继续查找对象是否带有 someKey 这个实例变量（iVar），如果还没有找到，程序会继续试图调用 -(id) valueForUndefinedKey:这个方法。如果这个方法还是没有被实现的话，程序会抛出一个 NSUndefinedKeyException 异常错误。
(Key-Value Coding 查找方法的时候，不仅仅会查找 someKey 这个方法，还会查找 getsomeKey 这个方法，前面加一个 get，或者_someKey 以及_getsomeKey 这几种形式。同时，查找实例变量的时候也会不仅仅查找 someKey 这个变量，也会查找_someKey 这个变量是否存在。)
设计 valueForUndefinedKey:方法的主要目的是当你使用-(id)valueForKey 方法从对象中请求值时，对象能够在错误发生前，有最后的机会响应这个请求。

45:进程死锁的原因

答案：资源竞争及进程推进顺序非法

死锁的 4 个必要条件

答案：互斥、请求保持、不可剥夺、环路

死锁的处理

答案：鸵鸟策略、预防策略、避免策略、检测与解除死锁

45.目标-动作机制

目标是动作消息的接收者。一个控件，或者更为常见的是它的单元，以插座变量（参见"插座变量"部分） 的形式保有其动作消息的目标。

动作是控件发送给目标的消息，或者从目标的角度看，它是目标为了响应动作而实现的方法。

程序需要某些机制来进行事件和指令的翻译。这个机制就是目标-动作机制。