内存管理

ARC处理原理

ARC是Objective-C编译器的特性，而不是运行时特性或者垃圾回收机制，ARC所做的只不过是在代码编译时为你自动在合适的位置插入release或autorelease，只要没有强指针指向对象，对象就会被释放。

- 前端编译器

前端编译器会为“拥有的”每一个对象插入相应的release语句。如果对象的所有权修饰符是__strong，那么它就是被拥有的。如果在某个方法内创建了一个对象，前端编译器会在方法末尾自动插入release语句以销毁它。而类拥有的对象（实例变量/属性）会在dealloc方法内被释放。事实上，你并不需要写dealloc方法或调用父类的dealloc方法，ARC会自动帮你完成一切。此外，由编译器生成的代码甚至会比你自己写的release语句的性能还要好，因为编辑器可以作出一些假设。在ARC中，没有类可以覆盖release方法，也没有调用它的必要。ARC会通过直接使用objc_release来优化调用过程。而对于retain也是同样的方法。ARC会调用objc_retain来取代保留消息。

- ARC优化器

虽然前端编译器听起来很厉害的样子，但代码中有时仍会出现几个对retain和release的重复调用。ARC优化器负责移除多余的retain和release语句，确保生成的代码运行速度高于手动引用计数的代码。

下面关于Objective-C内存管理的描述错误的是

```
A 当使用ARC来管理内存时，代码中不可以出现autorelease
B autoreleasepool 在 drain 的时候会释放在其中分配的对象
C 当使用ARC来管理内存时，在线程中大量分配对象而不用autoreleasepool则可能会造成内存泄露
D 在使用ARC的项目中不能使用NSZone
```

- 参考答案：A
- 理由：ARC只是在大多时候编译自动为我们添加上内存管理的代码，只是我们的源代码看不到而已，但是在编译时，编译器会添加上相关内存管理代码。对于自动释放池，在drain时会将自动释放池中的所有对象的引用计数减一，若引用计数为0，则会自动释放掉其内存。如果在线程中需要大量分配内存，我们理应添加上自动释放池，以防内存泄露。比如在for循环中要分配大量的内存处理数据，那么我们应该在for循环内添加自动释放池，在每个循环后就将内存释放掉，防止内存泄露。在ARC项目中，自然不能手动使用NSZone，也不能调用父类的dealloc。

MRC文件在ARC工程混合编译时，需要在文件的Compiler Flags上添加什么参数

```
A -shared
B -fno-objc-arc
C -fobjc-arc
D -dynamic
参考答案：B
```

什么情况使用 weak 关键字，相比 assign 有什么不同?

- 什么情况使用weak关键字?
- 在 ARC 中,在有可能出现循环引用的时候,往往要通过让其中一端使用 weak 来解决,比如:
  delegate 代理属性
  自身已经对它进行一次强引用,没有必要再强引用一次,此时也会使用 weak,自定义 IBOutlet 控件属性一般也使用 weak；当然，也可以使用strong。
- weak与assign的不同?
- weak 此特质表明该属性定义了一种“非拥有关系” (nonowning relationship)。为这种属性设置新值时，设置方法既不保留新值，也不释放旧值。此特质同assign类似， 然而在属性所指的对象遭到摧毁时，属性值也会清空(nil out)。 而 assign 的“设置方法”只会执行针对“纯量类型” (scalar type，例如 CGFloat 或 NSInteger 等)的简单赋值操作。
- assigin 可以用非 OC 对象,而 weak 必须用于 OC 对象

调用对象的release 方法会销毁对象吗?

```
    不会，调用对象的release 方法只是将对象的引用计数器-1，当对象的引用计数器为0的时候会调用了对象的dealloc 方法
```

自动释放池常见面试代码

```
for (int i = 0; i < someLargeNumber; ++i)
{
    NSString *string = @"Abc";
    string = [string lowercaseString];
    string = [string stringByAppendingString:@"xyz"];
    NSLog(@"%@",string);
}
```

问：以上代码存在什么样的问题？如果循环的次数非常大时，应该如何修改？

```
存在问题：问题处在每执行一次循环，就会有一个string加到当前runloop中的自动释放池中，只有当自动释放池被release的时

解决办法1：如果i比较大，可以用@autoreleasepool {}解决，放在for循环外，循环结束后，销毁创建的对象，解决占据栈区内

解决方法2：如果i玩命大，一次循环都会造成自动释放池被填满，自动释放池放在for循环内，每次循环都将上一次创建的对象re
```

```
修改之后:
for(int i = 0; i<1000;i++){
NSAutoreleasePool * pool1 = [[NSAutoreleasePool alloc] init];
NSString *string = @"Abc";
string = [string lowercaseString];
string = [string stringByAppendingString:@"xyz"];
NSLog(@"%@",string);
//释放池
[pool1 drain]; }
```

objective-C对象的内存布局是怎样的?

- 由于Objective-C中没有多继承，因此其内存布局还是很简单的，就是：最前面有个isa指针，然后父类的实例变量存放在子类的成员变量之前

看下面的程序,第一个NSLog会输出什么?这时str的retainCount是多少?第二个和第三个呢? 为什么?

```
NSMutableArray* ary = [[NSMutableArray array] retain];
NSString *str = [NSString stringWithFormat:@"test"];
[str retain];
[aryaddObject:str];
NSLog(@”%@%d”,str,[str retainCount]);
[str retain];
[str release];
[str release];
NSLog(@”%@%d”,str,[str retainCount]);
[aryremoveAllObjects]
NSLog(@”%@%d”,str,[str retainCount]);
str的retainCount创建+1，retain+1，加入数组自动+1 3
retain+1，release-1，release-1 2
数组删除所有对象，所有数组内的对象自动-1 1
```

回答person的retainCount值,并解释为什么

```
Person * per = [[Person alloc] init]; 此时person 的retainCount的值是1 self.person = per;
在self.person 时,如果是assign,person的 retainCount的值不变,仍为1 若是:retain person的retainCount的值加1,变为
若是:copy person的retainCount值不变,仍为1
```

什么时候需要在程序中创建内存池?

- 用户自己创建的数据线程，则需要创建该线程的内存池

如果我们不创建内存池，是否有内存池提供给我们?

- 界面线程维护着自己的内存池，用户自己创建的数据线程，则需要创建该线程的内存池

苹果是如何实现autoreleasepool的?

```
autoreleasepool以一个队列数组的形式实现,主要通过下列三个函数完成.
•   objc_autoreleasepoolPush
•   objc_autoreleasepoolPop
•   objc_autorelease
看函数名就可以知道，对autorelease分别执行push、pop操作。销毁对象时执行release操作。
```

objc使用什么机制管理对象内存?

- 通过引用计数器(retainCount)的机制来决定对象是否需要释放。 每次runloop完成一个循环的时候，都会检查对象的 retainCount，如果retainCount为0，说明该对象没有地方需要继续使用了，可以释放掉了。

为什么要进行内存管理?

- 因为移动设备的内存极其有限,当一个程序所占内存达到一定值时， 系统会发出内存警告. 当程序达到更大的值时, 程序会闪退, 影响用户体验. 为了保证程序的运行流畅, 必须进行内存管理

内存管理的范围?

- 管理所有继承自NSObject的对象, 对基本数据类型无效.是因为对象和其他数据类型在系统中存储的空间不一样,其他局部变量主要存储在栈区(因为基本数据类型占用的存储空间是固定的,一般存放于栈区),而对象存储于堆中,当代码块结束时,这个代码块所涉及到的所有局部变量会自动弹栈清空,指向对象的指针也会被回收,这时对象就没有指针指向,但依然存在于堆内存中,造成内存泄露.

objc使用什么机制管理对象内存(或者内存管理方式有哪些)？(重点)

- MRC(manual retain-release)手动内存管理
- ARC(automatic reference counting)自动引用计数
- Garbage collection (垃圾回收)。但是iOS不支持垃圾回收, ARC作为LLVM3.0编译器的一项特性, 在iOS5.0 (Xcode4) 版本后推出的。
- ARC的判断准则, 只要没有强指针指向对象, 对象就会被释放.

iOS是如何管理内存的?

- 这个问题的话上一个问题也提到过,讲下block的内存管理,ARC下的黄金法则就行。

- 这里说下swift里的内存管理:
  delgate照样weak修饰,闭包前面用[weak self],swift里的新东西,unowned,举例,如果self在闭包被调用的时候可能为空,则用weak,反之亦然,如果为空时使用了unowned,程序会崩溃,类似访问了悬挂指针,在oc中类似于unsafe_unretained,类似assign修饰了oc对象,对象被销毁后,被unowned修饰的对象不会为空,但是unowned访问速度更快,因为weak需要unwarp后才能使用

内存管理的原则

- 只要还有人在使用这个对象, 那么这个对象就不会被回收
- 只有你想使用这个对象, 那么就应该让这个对象的引用计数器加1
- 当你不想使用这个对象时, 应该让对象的引用计数器减1
- 谁创建, 就由谁来release
  - 如果你通过alloc, new, copy 来创建一个对象, 当你不想用这个对象的时候就必须调用release 或者autorelease 让引用计数器减1
  - 不是你创建的就不用你负责 release
- 谁retain 谁release
  - 只要你调用了retain ,无论这个对象如何生成, 都需要调用release
- 总结:
  有加就应该有减, 曾让某个计数器加1, 就应该让其在最后减1

内存管理研究的对象:

- 野指针:指针变量没有进行初始化或指向的空间已经被释放。
  - 使用野指针调用对象方法，会报异常，程序崩溃。
  - 通常再调用完release方法后，把保存对象指针的地址清空，赋值为nil，找oc中没有空指针异常，所以[nil retain]调用方法不会有异常。
- 内存泄露
  - 如 Person * person = [Person new]; (对象提前赋值nil或者清空)在栈区的person已经被释放, 而堆区new产生的对象还没有释放, 就会造成内存泄露
  - 在MRC手动引用计数器模式下, 造成内存泄露的情况
    1. 没有配对释放，不符合内存管理原则

2. 对象提前赋值nil或者清空，导致release不起作用。

- 僵尸对象：堆中已经被释放的对象(retainCount = 0)
- 空指针：指针赋值为空,nil

如何判断对象已经被销毁

- 重写dealloc方法，对象销毁时，会调用，重写时一定要[super dealloc]

retainCount = 0，使用retain能否复活对象

- 已经被释放的对象无法复活

对象与对象之间存在的关系

1. 继承关系
2. 组合关系（是一种强烈的包含关系）
3. 依赖关系(对象作为方法参数传递)

对象的组合关系中，确保成员变量不被提前释放?

- 重写set方法，在set方法中，retain该对象。

成员变量的对象，在哪里配对释放?

- dealloc中释放

对象组合关系中，内存泄露有哪几种情况?

- set方法没有retain对象
- 没有release旧对象
- 没有判断向set方法中传入的是否为同一个对象

正确重写set方法

- 判断是否为同一对象
- release旧对象
- retain新对象

分别描述内存管理要点、autorelease、release、NSAutoreleasePool?并说明autorelease是什么时候被release的?简述什么时候由你负责释放对象,什么时候不由你释放?[NSAutoreleasePool release]和[NSAutoreleasePool drain]有什么区别?

- 内存管理要点:Objective-C 使用引用计数机制(retainCount)来管理内存。
- 内存每被引用一次,该内存的引用计数+1,每被释放一次引 用计数-1。
- 当引用计数 = 0 的时候,调用该对象的 dealloc 方法,来彻底从内存中删除该对象。
- alloc,allocWithZone,new(带初始化)时:该对象引用计数 +1;
- retain:手动为该对象引用计数 +1;
- copy:对象引用计数 +1;//注意copy的OC数据类型是否有mutable，如有为深拷贝，新对象计数为1，如果没有，为浅拷贝，计数+1
- mutableCopy:生成一个新对象,新对象引用计数为 1;
- release:手动为该对象引用计数 -1;
- autorelease:把该对象放入自动释放池,当自动释放池释放时,其内的对象引用计数 -1。
- NSAutoreleasePool: NSAutoreleasePool是通过接收对象向它发送的autorelease消息,记录该对象的release消息,当自动释放池被销毁时,会自动向池中的对象发送release消息。
- autorelease 是在自动释放池被销毁,向池中的对象发送release
  只能释放自己拥有的对象。
- 区别是:在引用计数环境下(在不使用ARC情况下),两者基本一样,在GC(垃圾回收制)环境下,release是一个no-op(无效操作),所以无论是不是GC都使用drain
- 面试中内存管理,release和autorelease的含义?这里尤其要强调下autorelease,它引申出自动释放池,也能引申出Run loop!

自动释放池是什么,如何工作？

- 什么是自动释放池：用来存储多个对象类型的指针变量
- 自动释放池对池内对象的作用：存入池内的对象，当自动释放池被销毁时，会对池内对象全部做一次release操作
- 对象如何加入池中：调用对象的autorelease方法
- 自动释放池能嵌套使用吗：能
- 自动释放池何时被销毁 ： 简单的看，autorelease的"}"执行完以后。而实际情况是Autorelease对象是在当前的runloop迭代结束时释放的，而它能够释放的原因是系统在每个runloop迭代中都加入了自动释放池Push和Pop
- 多次调用对象的autorelease方法会导致：野指针异常
- 自动释放池的作用：将对象与自动释放池建立关系，池子内调用autorelease，在自动释放池销毁时销毁对象，延迟release销毁时间

自动释放池什么时候释放?

- 通过Observer监听RunLoop的状态，一旦监听到RunLoop即将进入睡眠等待状态，就释放自动释放池（kCFRunLoopBeforeWaiting）

IPhone OS有没有垃圾回收?autorelease 和垃圾回收制(gc)有什么关系?

- iOS 中没有垃圾回收。autorelease只是延迟释放,gc是每隔一段时间询问程序,看是否有无指针指向的对象,若有,就将它回收。他们两者没有什么关系。

ARC问题

1. 什么是arc机制：自动引用计数.
2. 系统判断对象是否销毁的依据：指向对象的强指针是否被销毁
3. arc的本质：对retainCount计算，创建+1 清空指针 - 1 或者到达autoreleasepool的大括号-1
4. arc目的：不需要程序员关心retain和release操作.
5. 如何解决arc机制下类的相互引用：.h文件中使用@class关键字声明一个类，两端不能都用强指针，一端用strong一端用weak

ARC通过什么方式帮助开发者管理内存?

ARC相对于MRC，不是在编译时添加retain/release/autorelease这么简单。应该是编译期和运行期两部分共同帮助开发者管理内存。

- 在编译期，ARC用的是更底层的C接口实现的retain/release/autorelease，这样做性能更好，也是为什么不能在ARC环境下手动retain/release/autorelease，同时对同一上下文的同一对象的成对retain/release操作进行优化（即忽略掉不必要的操作）

- ARC也包含运行期组件，这个地方做的优化比较复杂，但也不能被忽略，手动去做未必优化得好，因此直接交给编译器来优化，相信苹果吧！

开发项目时你是怎么检查内存泄露

- 静态分析 analyze
- instruments工具里面有个leak 可以动态分析

如果在block中多次使用 weakSelf的话，可以在block中先使用strongSelf，防止block执行时weakSelf被意外释放
对于非ARC，将 __weak 改用为 __block 即可

麻烦你设计个简单的图片内存缓存器（移除策略是一定要说的）

- 内存缓存是个通用话题，每个平台都会涉及到。cache算法会影响到整个app的表现。候选人最好能谈下自己都了解哪些cache策略及各自的特点。
- 常见的有FIFO,LRU,LFU等等。由于NSCache的缓存策略不透明，一些app开发者会选择自己做一套cache机制，其实并不难。
- FIFO : 新访问的数据插入FIFO队列尾部，数据在FIFO队列中顺序移动；淘汰FIFO队列头部的数据；
- LRU : 新数据插入到链表头部；每当缓存数据命中，则将数据移到链表头部；当链表满的时候，将链表尾部的数据丢弃；
- LFU : 新加入数据插入到队列尾部（因为引用计数为1）；队列中的数据被访问后，引用计数增加，队列重新排序；当需要淘汰数据时，将已经排序的列表最后的数据块删除；

常见的出现内存循环引用的场景有哪些?

- 定时器（NSTimer）：NSTimer经常会被作为某个类的成员变量，而NSTimer初始化时要指定self为target，容易造成循环引用（self->timer->self）。 另外，若timer一直处于validate的状态，则其引用计数将始终大于0，因此在不再使用定时器以后，应该先调用invalidate方法
- block的使用：block在copy时都会对block内部用到的对象进行强引用(ARC)或者retainCount增1(非ARC)。在ARC与非ARC环境下对block使用不当都会引起循环引用问题， 一般表现为，某个类将block作为自己的属性变量，然后该类在block的方法体里面又使用了该类本身，简单说就是self.someBlock =Type var{[self dosomething];或者self.otherVar = XXX;或者_otherVar = …};出现循环的原因是：self->block->self或者self->block->_ivar（成员变量）
- 代理（delegate）：在委托问题上出现循环引用问题已经是老生常谈了，规避该问题的杀手锏也是简单到哭，一字诀：声明delegate时请用assign(MRC)或者weak(ARC)，千万别手贱玩一下retain或者strong，毕竟这基本逃不掉循环引用了！

对象添加到通知中心中，当通知中心发通知时，这个对象却已经被释放了，可能会出现什么问题?

- 其实这种只是考查对通知的简单应用。通知是多对多的关系，主要使用场景是跨模块传值。当某对象加入到通知中心后，若在对象被销毁前不将该对象从通知中心中移除，当发送通知时，就会造成崩溃。这是很常见的。所以，在添加到通知中心后，一定要在释放前移除。

ARC下不显式指定任何属性关键字时，默认的关键字都有哪些?

- 对于基本数据类型默认关键字是：atomic,readwrite,assign
- 对于普通的Objective-C对象：atomic,readwrite,strong

写一个便利构造器

```
+(id)Person {
Person *person=[Person alloc]init];
return [person autorelease]; 备注:ARC时不用 autorelease
}
```

写出下面程序段的输出结果

```
NSDictionary *dict = [NSDictionary dictionaryWithObject:@"a string value" forKey:@"akey"]; NSLog(@"%@",
[dict release];
打印输出 a string value,然后崩溃----原因:便利构造器创建的对象,之后的release,会造成过度释放
```

请写出以下代码的执行结果

```
NSString * name = [ [ NSString alloc] init ];
name = @”Habb”;
[ name release];
打印输出结果是: Habb,在[name release]前后打印均有输出结果 ---会造成内存泄露---原先指向的区域变成了野指针,之后的
```

写出方法获取ios内存使用情况?

iOS是如何管理内存的?

我相信很多人的回答是内存管理的黄金法则，其实如果我是面试官，我想要的答案不是这样的。我希望的回答是工作中如何处理内存管理的。

参考答案：

- Block内存管理：由于使用block很容易造成循环引用，因此一定要小心内存管理问题。最好在基类controller下重写dealloc，加一句打印日志，表示类可以得到释放。如果出现无打印信息，说明这个类一直得不到释放，表明很有可能是使用block的地方出现循环引用了。对于block中需要引用外部controller的属性或者成员变量时，一定要使用弱引用，特别是成员变量像_testId这样的，很多人都没有使用弱引用，导致内存得不到释放。
- 对于普通所创建的对象，因为现在都是ARC项目，所以记住内存管理的黄金法则就可以解决。

很多内置的类，如tableview的delegate的属性是assign不是retain?

- tableview的代理一般都是它所属的控制器，控制器会对它内部的view进行一次retain操作，而tableview对代理控制器也进行一次retain操作，就会出现循环引用问题。