

- 面试题出自技术群分享
- 欢迎转载，转载请注明出处：[pmst-swiftgg](#)
- 调试好可运行的源码 [objc-runtime](#)，官网找 [objc4](#)
- 完成度：0%
- 最后修订：2020/05/24

一面

1. 你们 hybrid APP 技术方案跨端方案(weex、RN、flutter)选择? 与 h5 hybrid 方案相比优缺点, 哪些场景需要用到 native、哪些页面 h5、哪些页面用跨端方案(weex、RN、flutter)?

2. 如果你做的是一个超级 APP (微信、淘宝), 里面有一个引擎可以运行不同的小程序, 你如何设计保证小程序之间的安全性?

3. 介绍一下你们 APP 的架构设计

4. H5 与 跨端方案(weex、RN、flutter) 如何进行代码复用、适配?

5. 进程和线程的区别? (不说到 CPU 调度, 会一直追问)

进程和线程的区别

6. OC 是否支持重载? 为什么?

7. 分类是否能够添加属性

关联对象基本使用方法:

```
#import <objc/runtime.h>

static NSString * const kKeyOfImageProperty;

@implementation UIView (Image)

- (UIImage *)pt_image {
    return objc_getAssociatedObject(self, &kKeyOfImageProperty);
}

- (void)setPTImage:(UIImage *)image {
    objc_setAssociatedObject(self, &kKeyOfImageProperty, image, OBJC_ASSOCIATION_RETAIN);
}

@end
```

更多问题:

- 关联对象需要手动处理生命周期吗? 释放时机是什么时候?
- 关联对象的如何进行内存管理的? 关联对象如何实现 weak 属性?
 - 使用了 policy 设置内存管理策略, 具体见上。凡是加一层呗。。。搞一个类, 然后内部封装一个 weak 变量持有; 或者不用 weak, 但是还是封装一层, 但是在 dealloc 中进行置为 nil 操作

8. OC 内存泄漏的原因

1. 存在 retainCycle 循环引用导致对象相互持有无法释放造成内存泄露;
 1. Block, 内部使用了 self 或直接 _instance 访问实例变量也会隐式捕获 self;
 2. delegate 循环引用, 比如使用 strong 关键字去修饰;
 3. NSTimer 和 CADisplayLink, 两者内存泄露原因差不多, 以 NSTimer 为例, 这里 timer 会强持有 target (通常就是 self 调用者本身), 然后 self 会用一个属性去持有 timer, 这里就形成了 retainCycle, 另外 timer 一旦加入到 runloop 就会被持有, 直至被 invalidated 才会被移除; 考虑 **repeats = NO** 的情况, 一旦触发过一次 timer 就会 invalidated, 也就会从 runloop 中移除。

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)t  
i target:(id)aTarget selector:(SEL)aSelector userInfo:(nullable  
id)userInfo repeats:(BOOL)yesOrNo;
```

2. 单例缓存视图或其他对象
3. 非 OC 对象内存处理, 比如:

```
CGImageRef ref = [context createCGImage:outputImage fromRect:output  
Image.extent];  
UIImage *endImg = [UIImage imageWithCGImage:ref];  
_imageView.image = endImg;  
CGImageRelease(ref); // 非 OC 对象需要手动内存释放
```

// 其他一

```
Ivar *ivars = class_copyIvarList(cls, &count);  
free(ivars);
```

// 其他二、

```
CFUUIDRef puuid = CFUUIDCreate( kCFAllocatorDefault );  
CFStringRef uuidString = CFUUIDCreateString( kCFAllocatorDefault, p  
uuid );  
NSString *uuid = [(NSString *)CFBridgingRelease(CFStringCreateCopy  
(NULL, uuidString)) uppercaseString];
```

```
// 使用完后释放 puuid 和 uuidString 对象
CFRelease(puuid);
CFRelease(uuidString);
```

9. 堆和栈的区别是什么?

link:堆和栈的区别

一、预备知识—程序的内存分配

一个由 C/C++ 编译的程序占用的内存分为以下几个部分

1. 栈区 (stack) — 由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。
2. 堆区 (heap) — 一般由程序员分配释放，若程序员不释放，程序结束时可能由 OS 回收。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。
3. 全局区 (静态区) (static) —，全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。- 程序结束后由系统释放。
4. 文字常量区 — 常量字符串就是放在这里的。程序结束后由系统释放
5. 程序代码区 — 存放函数体的二进制代码;

```
//main.cpp
int a = 0; //全局初始化区
char *p1; //全局未初始化区
main()
{
    int b; //栈
    char s[] = "abc"; //栈
    char *p2; //栈
    char *p3 = "123456"; //123456/0 在常量区, p3 在栈上。
    static int c = 0; //全局 (静态) 初始化区
    p1 = (char *)malloc(10);
    p2 = (char *)malloc(20);
    //分配得来 10 和 20 字节的区域就在堆区。
    strcpy(p1, "123456"); // 123456/0 放在常量区, 编译器可能会将它与 p3
    所指向的"123456"
    优化成一个地方。
}
```

二、堆和栈的理论知识

2.1 申请方式

stack: 由系统自动分配。例如，声明在函数中一个局部变量 `int b;` 系统自动在栈中为 `b` 开辟空间 **heap:** 需要程序员自己申请，并指明大小，在 `c` 中 `malloc` 函数如 `p1`

= (char *)malloc(10) 在 C++ 中用 new 运算符如 p2 = new char[10] 但是注意 p1、p2 本身是在栈中的。

2.2 申请后系统的响应

栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。堆：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样，代码中的 delete 语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

2.3 申请大小的限制

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用的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

2.4 申请效率的比较：

栈由系统自动分配，速度较快。但程序员是无法控制的。

堆是由 new 分配的内存，一般速度比较慢，而且容易产生内存碎片，不过用起来最方便。

另外，在 WINDOWS 下，最好的方式是用 VirtualAlloc 分配内存，他不是堆，也不是在栈是直接在进程的地址空间中保留一块内存，虽然用起来最不方便。但是速度快，也最灵活。

2.5 堆和栈中的存储内容

栈：在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的 C 编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容由程序员安排。

2.6 存取效率的比较

```
char s1[] = "aaaaaaaaaaaaaaaa";  
char *s2 = "bbbbbbbbbbbbbbbb";  
aaaaaaaaaaaa 是在运行时刻赋值的；
```

而 bbbbbbbbbbbb 是在编译时就确定的；
但是，在以后的存取中，在栈上的数组比指针所指向的字符串（例如堆）快。
比如：

```
#include
void main()
{
    char a = 1;
    char c[] = "1234567890";
    char *p = "1234567890";
    a = c[1];
    a = p[1];
    return;
}
```

对应的汇编代码

```
10: a = c[1];
00401067 8A 4D F1 mov cl,byte ptr [ebp-0Fh]
0040106A 88 4D FC mov byte ptr [ebp-4],cl
11: a = p[1];
0040106D 8B 55 EC mov edx,dword ptr [ebp-14h]
00401070 8A 42 01 mov al,byte ptr [edx+1]
00401073 88 45 FC mov byte ptr [ebp-4],al
```

第一种在读取时直接就把字符串中的元素读到寄存器 cl 中，而第二种则要先把指针值读到

edx 中，再根据 edx 读取字符，显然慢了。

2.7 小结:

堆和栈的区别可以用如下的比喻来看出：使用栈就象我们去饭馆里吃饭，只管点菜（发出申请）、付钱、和吃（使用），吃饱了就走，不必理会切菜、洗菜等准备工作和洗碗、刷锅等扫尾工作，他的好处是快捷，但是自由度小。使用堆就象是自己动手做喜欢吃的菜肴，比较麻烦，但是比较符合自己的口味，而且自由大。（经典！）

10. 栈、堆分别是否会被线程所共享？

11. 内存空间中除了堆和栈还有什么内容？

1. 栈区（stack）— 由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。
2. 堆区（heap）— 一般由程序员分配释放，若程序员不释放，程序结束时可能由 OS 回收。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。
3. 全局区（静态区）（static）—，全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。- 程序结束后由系统释放。
4. 文字常量区 — 常量字符串就是放在这里的。程序结束后由系统释放

5. 程序代码区—存放函数体的二进制代码;

12. 讲一下你做过的性能优化?

13. HTTP 请求方法种类有哪些?(别忘记 HEAD)

- GET: 幂等性、安全性、可缓存性
- POST
- PUT
- DELETE
- HEAD

14. 如何实现一个网络监控? -追问-> 如果拦截 NSURLSession 具体会怎么做? 涉及哪些方法,方法名字说下?如何保证线程安全(并发场景下, 监控的线程和正常业务方的线程, 如何保证正常回调?) (参考 Notification 的回调逻辑)

可查看 FLEX 源码

15. Swift、OC 如何相互调用? Swift->OC、OC ->Swift? 我开发一个 Swift 的 SDK,(API 都是 Swift 的), 内部需要调用到 OC 的库, 要怎么做?.

16. 动态库和静态库区别, 优缺点辨析? -追问->包大小的差异的原因?为什么会有差异?

Q&A 八问: 静态库和动态库、armv7,armv7s,i386,x86_64 架构