内存管理

@M了个J

https://github.com/CoderMJLee

实力IT教育 www.520it.com

小码哥教育 SEEMYGO

面试题

- 使用CADisplayLink、NSTimer有什么注意点？
- 介绍下内存的几大区域
- 讲一下你对 iOS 内存管理的理解
- ARC 都帮我们做了什么？
  - LLVM + Runtime
- weak指针的实现原理
- autorelease对象在什么时机会被调用release
- 方法里有局部对象， 出了方法后会立即释放吗

CADisplayLink、NSTimer使用注意

- CADisplayLink、NSTimer会对target产生强引用，如果target又对它们产生强引用，那么就会引发循环引用

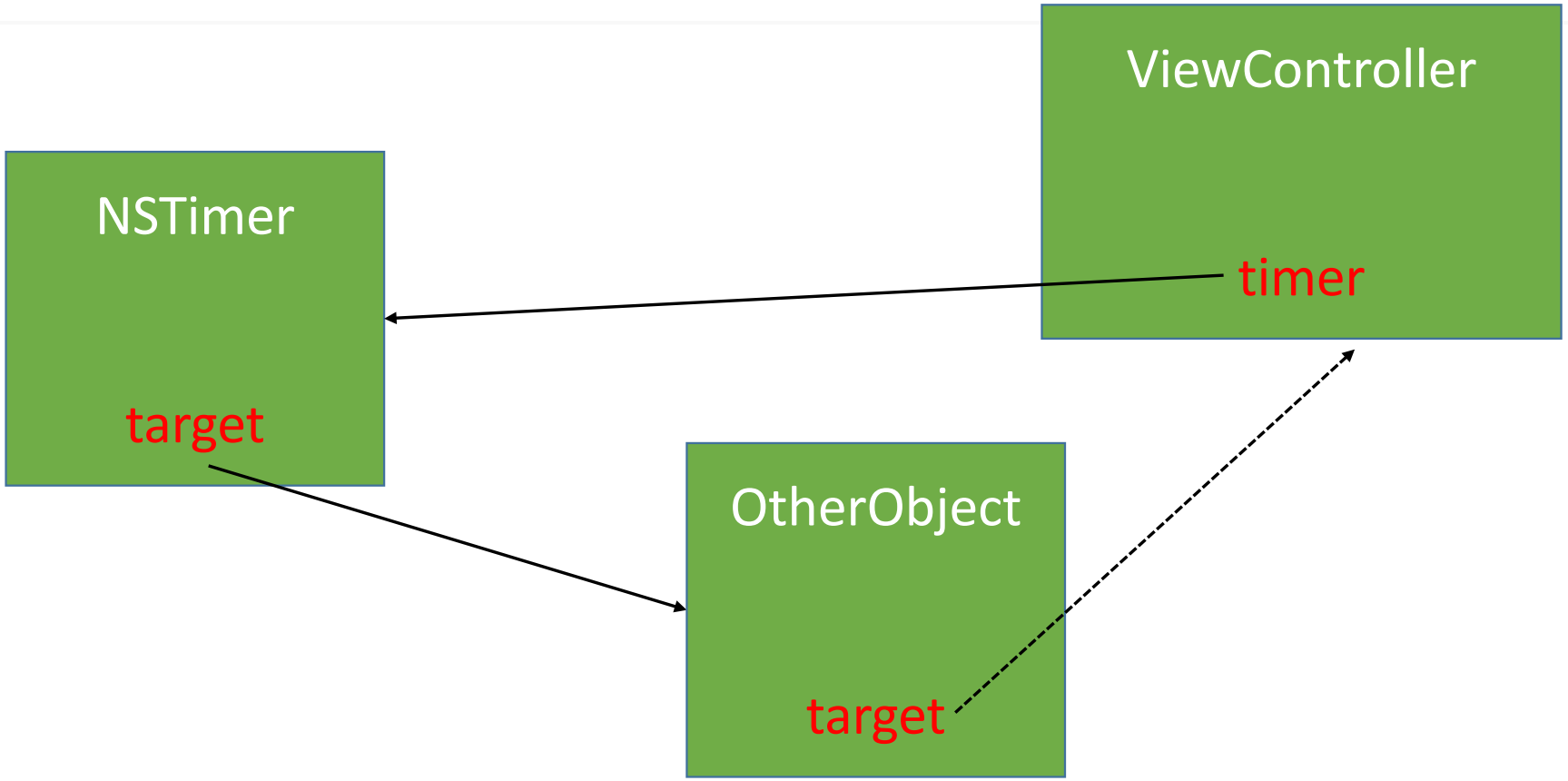
- 解决方案
  - 使用block

```
__weak typeof(self) waekSelf = self;
self.timer = [NSTimer scheduledTimerWithTimeInterval:2.0
                                              repeats:YES
                                                block:^(NSTimer * _Nonnull timer) {
    [waekSelf test];
}];
```

  - 使用代理对象（NSProxy）

```
+ (instancetype)proxyWithTarget:(id)target
{
    MJTimerProxy *proxy = [self alloc];
    proxy.target = target;
    return proxy;
}

- (NSMethodSignature *)methodSignatureForSelector:(SEL)sel
{
    return [self.target methodSignatureForSelector:sel];
}

- (void)forwardInvocation:(NSInvocation *)invocation
{
    [invocation invokeWithTarget:self.target];
}
```

```
MJTimerProxy *proxy = [MJTimerProxy proxyWithTarget:self];
self.timer = [NSTimer scheduledTimerWithTimeInterval:2.0
                                              target:proxy
                                            selector:@selector(test)
                                            userInfo:nil
                                             repeats:YES];
```

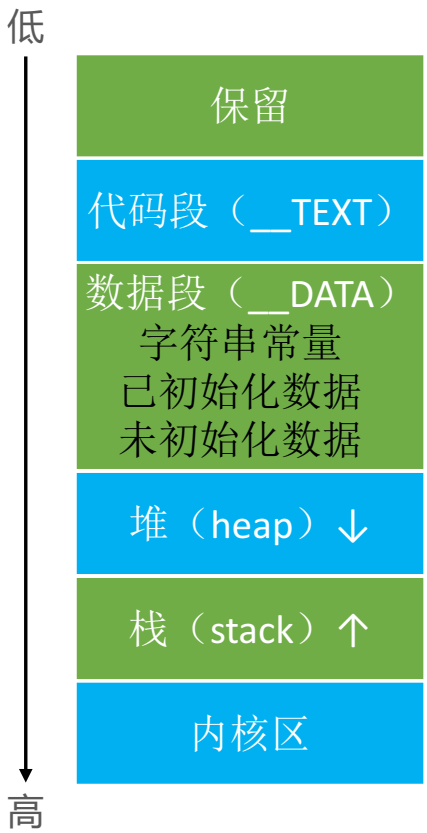

ViewController
timer
NSTimer
target
OtherObject
target

GCD定时器

- NSTimer依赖于RunLoop，如果RunLoop的任务过于繁重，可能会导致NSTimer不准时
- 而GCD的定时器会更加准时

```
// 创建一个定时器
dispatch_source_t timer = dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER, 0, 0, queue);
// 设置时间（start是几秒后开始执行，interval是时间间隔）
dispatch_source_set_timer(timer,
                          dispatch_time(DISPATCH_TIME_NOW, (int64_t)(start * NSEC_PER_SEC)),
                          (uint64_t)(interval *NSEC_PER_SEC),
                          0);
// 设置回调
dispatch_source_set_event_handler(timer, ^{

});
// 启动定时器
dispatch_resume(timer);
```

iOS程序的内存布局



- 代码段：编译之后的代码

- 数据段
  - 字符串常量：比如NSString *str = @"123"
  - 已初始化数据：已初始化的全局变量、静态变量等
  - 未初始化数据：未初始化的全局变量、静态变量等

- 栈：函数调用开销，比如局部变量。分配的内存空间地址越来越小

- 堆：通过alloc、malloc、calloc等动态分配的空间，分配的内存空间地址越来越大

小码哥教育 SEEMYGO

Tagged Pointer

- 从64bit开始，iOS引入了Tagged Pointer技术，用于优化NSNumber、NSDate、NSString等小对象的存储
- 在没有使用Tagged Pointer之前， NSNumber等对象需要动态分配内存、维护引用计数等，NSNumber指针存储的是堆中NSNumber对象的地址值
- 使用Tagged Pointer之后，NSNumber指针里面存储的数据变成了：Tag + Data，也就是将数据直接存储在了指针中
- 当指针不够存储数据时，才会使用动态分配内存的方式来存储数据
- objc_msgSend能识别Tagged Pointer，比如NSNumber的intValue方法，直接从指针提取数据，节省了以前的调用开销
- 如何判断一个指针是否为Tagged Pointer？
  - iOS平台，最高有效位是1（第64bit）
  - Mac平台，最低有效位是1

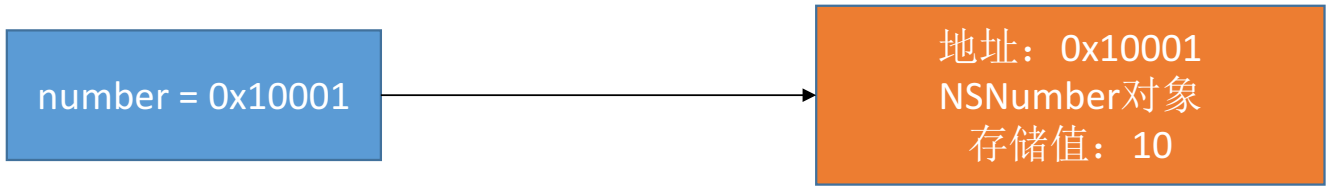
小码哥教育 SEEMYGO

判断是否为Tagged Pointer

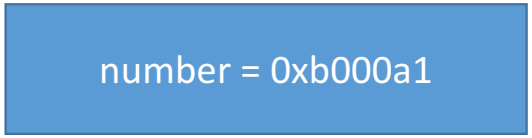
```
#if TARGET_OS_OSX && __x86_64__
    // 64-bit Mac - tag bit is LSB
#   define OBJC_MSB_TAGGED_POINTERS 0
#else
    // Everything else - tag bit is MSB
#   define OBJC_MSB_TAGGED_POINTERS 1
#endif

#if OBJC_MSB_TAGGED_POINTERS
#   define _OBJC_TAG_MASK (1UL<<63)
#else
#   define _OBJC_TAG_MASK 1UL
#endif
```

```
static inline bool
_objc_isTaggedPointer(const void * _Nullable ptr)
{
    return ((uintptr_t)ptr & _OBJC_TAG_MASK) == _OBJC_TAG_MASK;
}
```

使用Tagged Pointer之前



使用Tagged Pointer之后



小码哥教育 SEEMYGO

面试题

- 思考以下2段代码能发生什么事？有什么区别？

```
dispatch_queue_t queue = dispatch_get_global_queue(0, 0);
for (int i = 0; i < 1000; i++) {
    dispatch_async(queue, ^{
        self.name = [NSString stringWithFormat:@"abcdefghijk"];
    });
}
```

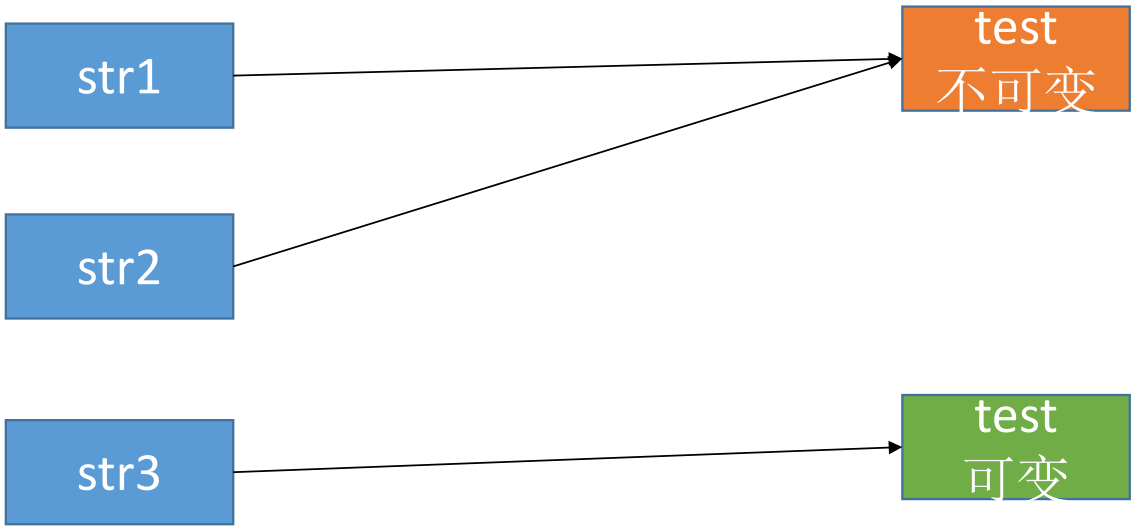
```
dispatch_queue_t queue = dispatch_get_global_queue(0, 0);
for (int i = 0; i < 1000; i++) {
    dispatch_async(queue, ^{
        self.name = [NSString stringWithFormat:@"abc"];
    });
}
```

OC对象的内存管理

- 在iOS中，使用引用计数来管理OC对象的内存
- 一个新创建的OC对象引用计数默认是1，当引用计数减为0，OC对象就会销毁，释放其占用的内存空间
- 调用retain会让OC对象的引用计数+1，调用release会让OC对象的引用计数-1
- 内存管理的经验总结
  - 当调用alloc、new、copy、mutableCopy方法返回了一个对象，在不需要这个对象时，要调用release或者autorelease来释放它
  - 想拥有某个对象，就让它的引用计数+1；不想再拥有某个对象，就让它的引用计数-1
- 可以通过以下私有函数来查看自动释放池的情况
  - `extern void _objc_autoreleasePoolPrint(void);`

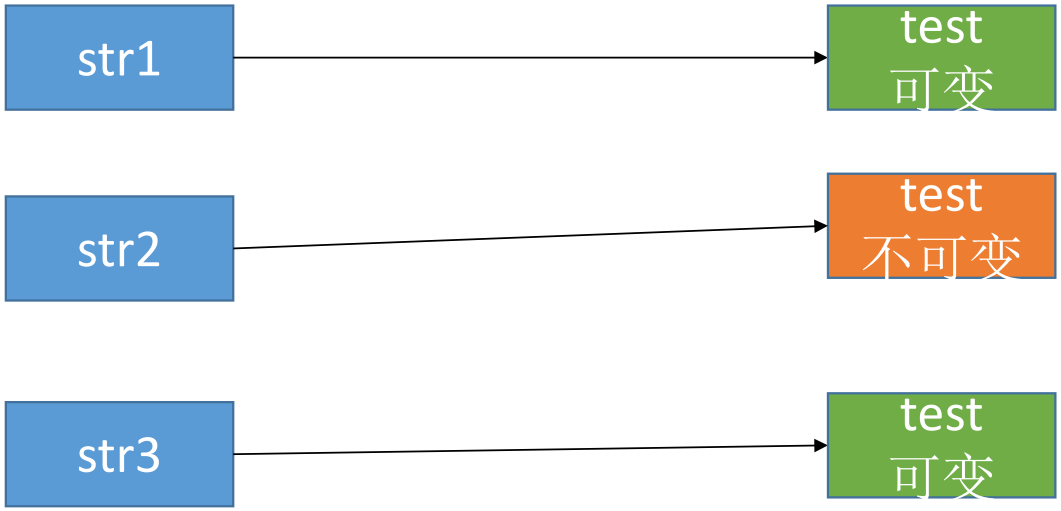
小码哥教育 SEEMYGO

```objc
NSString *str1 = [[NSString alloc] initWithFormat:@"test"];
NSString *str2 = [str1 copy];
NSMutableString *str3 = [str1 mutableCopy];

[str3 release];
[str2 release];
[str1 release];
```



小码哥教育 SEEMYGO

```objc
NSMutableString *str1 = [[NSMutableString alloc] initWithFormat:@"test"];
NSString *str2 = [str1 copy];
NSMutableString *str3 = [str1 mutableCopy];

[str1 release];
[str2 release];
[str3 release];
```



小码哥教育 SEEMYGO

copy和mutableCopy

	copy	mutableCopy
NSString	NSString 浅拷贝	NSMutableString 深拷贝
NSMutableString	NSString 深拷贝	NSMutableString 深拷贝
NSArray	NSArray 浅拷贝	NSMutableArray 深拷贝
NSMutableArray	NSArray 深拷贝	NSMutableArray 深拷贝
NSDictionary	NSDictionary 浅拷贝	NSMutableDictionary 深拷贝
NSMutableDictionary	NSDictionary 深拷贝	NSMutableDictionary 深拷贝

小码哥教育 SEEMYGO

引用计数的存储

- 在64bit中，引用计数可以直接存储在优化过的isa指针中，也可能存储在SideTable类中

```
struct SideTable {
    spinlock_t slock;
    RefcountMap refcnts;
    weak_table_t weak_table;
};
```

- refcnts是一个存放着对象引用计数的散列表

dealloc

- 当一个对象要释放时，会自动调用dealloc，接下的调用轨迹是
- dealloc
- _objc_rootDealloc
- rootDealloc
- object_dispose
- objc_destructInstance、free

```
void *objc_destructInstance(id obj)
{
    if (obj) {
        // Read all of the flags at once for performance.
        bool cxx = obj->hasCxxDtor();
        bool assoc = obj->hasAssociatedObjects();

        // This order is important.
        if (cxx) object_cxxDestruct(obj); // 清除成员变量
        if (assoc) _object_remove_assocations(obj);
        obj->clearDeallocating(); // 将指向当前对象的弱指针置为nil
    }

    return obj;
}
```

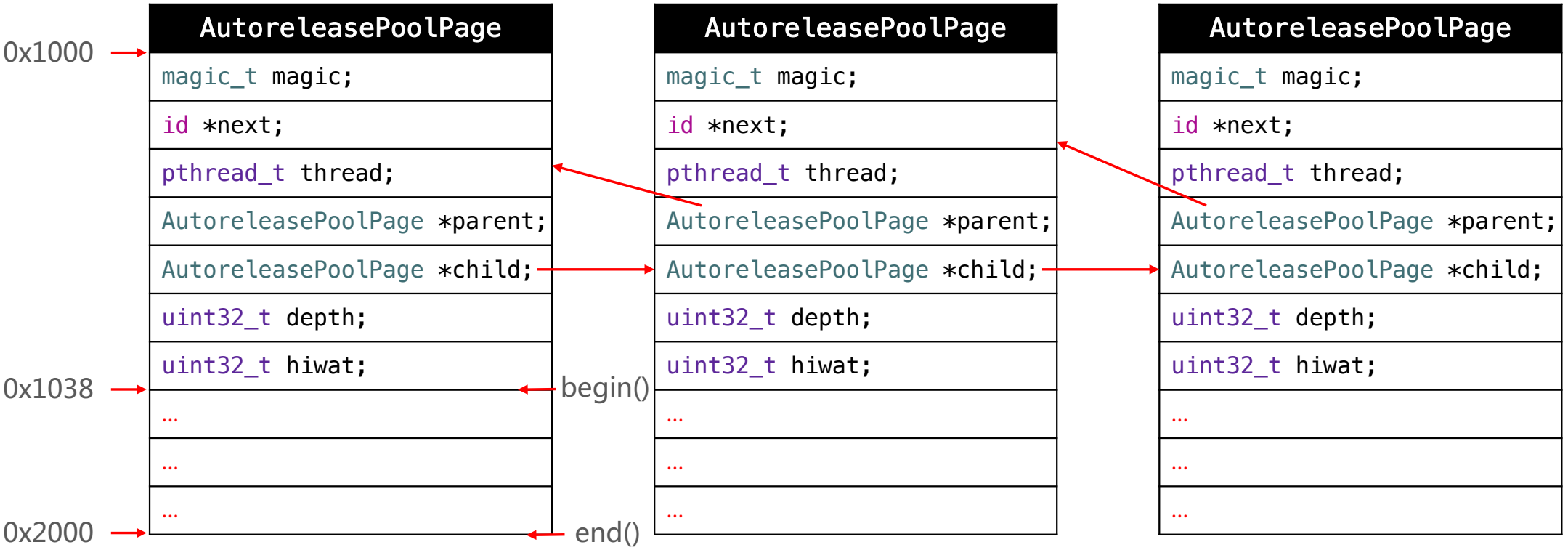
自动释放池

- 自动释放池的主要底层数据结构是：__AtAutoreleasePool、AutoreleasePoolPage
- 调用了autorelease的对象最终都是通过AutoreleasePoolPage对象来管理的
- 源码分析
  - clang重写@autoreleasepool
  - objc4源码：NSObject.mm

```
class AutoreleasePoolPage
{
    magic_t const magic;
    id *next;
    pthread_t const thread;
    AutoreleasePoolPage * const parent;
    AutoreleasePoolPage *child;
    uint32_t const depth;
    uint32_t hiwat;
}
```

AutoreleasePoolPage的结构

- 每个AutoreleasePoolPage对象占用4096字节内存，除了用来存放它内部的成员变量，剩下的空间用来存放autorelease对象的地址
- 所有的AutoreleasePoolPage对象通过双向链表的形式连接在一起



小码哥教育 SEEMYGO

AutoreleasePoolPage的结构

- 调用push方法会将一个POOL_BOUNDARY入栈，并且返回其存放的内存地址
- 调用pop方法时传入一个POOL_BOUNDARY的内存地址，会从最后一个入栈的对象开始发送release消息，直到遇到这个POOL_BOUNDARY
- id *next指向了下一个能存放autorelease对象地址的区域

Runloop和Autorelease

- iOS在主线程的Runloop中注册了2个Observer
- 第1个Observer监听了`kCFRunLoopEntry`事件，会调用`objc_autoreleasePoolPush()`
- 第2个Observer
- ✓ 监听了`kCFRunLoopBeforeWaiting`事件，会调用`objc_autoreleasePoolPop()`、`objc_autoreleasePoolPush()`
- ✓ 监听了`kCFRunLoopBeforeExit`事件，会调用`objc_autoreleasePoolPop()`