第23节课内容总结

内存5大区

- 堆：存放通过`alloc`，`new`，`malloc`等创建出来的变量。`一般以0x6开头`
- 栈区：存放局部变量，方法参数，`对象的指针`等。`栈的内存大小有限，主线程1MB，其他线程512KB`，栈是线程独有的，栈在线程开始的时候初始化，每个线程的栈互相独立`一般以0x7开头`
- 全局区/静态区：存放全局变量和静态变量。`一般以0x1开头`，编译时分配的内存区域在程序运行时一直存在，直到程序运行结束才会被释放
- 常量区：常量。(const修饰)`一般以0x1开头`，常量区的内存和全局区一样也是在编译阶段进行分配，程序运行时会一直存储在内存中，只有当程序结束后才会由操作系统释放
- 代码区：存放编译生成的二进制代码

引用计数

引用计数（Reference Count）是一个简单而有效的管理对象生命周期的方式。

iOS中的内存管理方案

`NONPOINTER_ISA`，`Tagged Pointer`，`SideTable`

Tagged Pointer

`Tagged Ponter`是苹果在64位操作系统下提出来的概念，也就是从5S开始。`Tagged Ponter`针对的是小对象类型，比如NSNumber、NSDate、NSString。

`Tagged Pointer`也是指针，是一种被打上了tagged标记的指针。

`Tagged Pointer`指针，它表示的不再是地址，而是真正的值。这样能够大幅度的提升它的访问速度和创建销毁的速度。因为在栈上，不必在堆上为其分配内存，节省了很多内存开销。在性能上，根据苹果官方的说法，`Tagged Pointer`有3倍的空间效率的提升，以及106倍的创建和销毁速度的提升。

retain流程

- 判断是否是`Tagged Pointer`，`Tagged Pointer`不需要维护引用计数，直接返回

- 如果不是`Tagged Pointer`，获取对象的`isa`，然后判断是否是`NONPOINTER_ISA`
- 如果不是`NONPOINTER_ISA`，交给散列表处理，对其引用计数进行`++`操作，然后返回
- 判断是否正在析构，如果是直接返回
- 如果是`NONPOINTER_ISA`，对`isa`的`extra_rc`进行`++`操作
- 如果超出了`extra_rc`的最大存储范围，就将一半的引用计数保存在`extra_rc`，并且把`isa`的`has_sidetable_rc`置为1，然后将另一半的引用计数保存到散列表

当引用计数超出isa的extra_rc的最大存储范围时，为什么要extra_rc和散列表中各存一半呢，为什么不是把所有的引用计数的值都存到散列表里面?

通过isa可以很容易的拿到extra_rc，通过extra_rc进行引用计数的存储是很方便的。散列表是先拿到SideTable这张表，再在表中拿到引用计数表，才能进行操作，表操作还要做加锁和解锁操作，非常浪费性能。所以在SideTable存一半，这样的话++，--都能够在extra_rc里面对引用计数进行操作，效率能够更高。

release流程

- 判断是否是`Tagged Pointer`，`Tagged Pointer`不需要维护引用计数，直接返回
- 如果不是`Tagged Pointer`，获取对象的`isa`，然后判断是否是`NONPOINTER_ISA`
- 如果不是`NONPOINTER_ISA`，交给散列表处理，对其引用计数进行`--`操作，如果散列表的引用计数清零，对该对象执行`dealloc`操作，然后返回
- 如果是`NONPOINTER_ISA`，对`isa`的`extra_rc`进行`--`操作，当`extra_rc`计数为0，则需要向散列表的引用计数借位
- 判断`isa`的`has_sidetable_rc`是否为1，如果不为1，对该对象执行`dealloc`操作。
- 如果`isa`的`has_sidetable_rc`为1，获取散列表的引用计数，如果散列表的引用计数为0，对该对象执行`dealloc`操作
- 如果散列表的引用计数大于0，将引用计数-1，然后存入`isa`的`extra_rc`

dealloc流程

- 判断是否是`Tagged Pointer`，`Tagged Pointer`不需要维护引用计数，直接返回
- 如果是`NONPOINTER_ISA`，且没有弱引用，没有关联对象，没有c++析构函数，没有散列表引用计数，直接释放，否则进行下一步
- 调用`objc_dispose()`函数
- 调用`objc_destructInstance()`函数，然后再调用`free()`函数释放
- `objc_destructInstance`函数首先判断是否存在c++析构函数和是否存在关联对象，如果有

则调用c++析构函数、删除关联对象，然后清除弱引用表的相关信息和清除散列表的相关信息