内存管理基础

概念

创建oc对象，定义变量，调用函数或方法都会占用有限的内存，当app占用的内存过多时，就需要回收一些不再需要使用的内存空间，否则造成闪退会影响用户体验。

- 涉及操作
  - 分配内存
  - 回收内存
- 管理范围
  - 任何继承了NSObject的对象
  - 对其他非对象类型（int float char等等）无效
- 本质原因
  - oc对象存放在堆
  - 非oc对象**一般**存放在栈

```
int main(){
  @autoreleasepool{
          int a=10,b=20;
          Person *p = [[Person alloc] init];
  }return 0;
}
//a b p都在栈，代码结束后会自动回收，而Person对象计数器为1，还留在内存中
```

引用计数器

根据对象的引用计数器来判断什么时候需要回收一个对象所占用的内存

- 每一个oc对象都有自己的引用计数器
- 是一个整数
- 可以理解为对象被引用的次数，有多少个人正在用这个对象
- 当引用计数器为0时，对象占用的内存就会被系统回收
- 如果在程序整个运行过程中，引用计数器都不为0，那么它会一直在内存中
- 初始值（被初始化的时候）为1
- 常见操作
  - 给对象发送一条retain消息，引用计数器+1
  - 发送一条release消息，引用计数器-1
  - 发送retainCount消息，可以获得当前引用计数器值

dealloc基本概念

- 对象即将被销毁时，系统会自动给对象发送一条dealloc消息
- dealloc方法可以重写，一旦重写了dealloc方法，就必须调用[super dealloc]，并放在最后调用
- 不能直接调用，还要注意野指针问题
- 为了避免野指针错误的常见方法：在对象被销毁后，将指向对象的指针变成空指针

autorelease基本使用

- autorelease是一种支持引用计数的内存管理方式，只要给对象发送一条autorelease消息，就会把对象放到一个自动释放池里，等到自动释放池销毁时，才会对池子里的所有对象做一次release操作
- 好处
  - 不用关心对象释放的时间
  - 不用关心什么时候调用release
- 原理
  - autorelease实际上是把对release的调用延迟了

```
Person *p = [person new];
p = [p autorelease];
//ios 5.0后
@autoreleasepool
{ // 创建自动释放池
} // 销毁自动释放池
NSAutoreleasePool *autoreleasePool = [[NSAutoreleasePool alloc] init];
Person *p = [[[Person alloc] init] autorelease];
[autoreleasePool drain];// 排水
@autoreleasepool{
  Person *p = [[Person new] autorelease];
}
```

Tips

- 并不是放到自动释放池代码里就会自动加入到自动释放池
- 要在自动释放池代码里调用autorelease才有效
- 自动释放池不适合放入占用内存比较大的对象
- 不要把大量循环放到同一个autoreleasepool中，这样会造成内存峰值上升
- 使用autorelease不要过度释放

ARC

- Automatic Reference Counting，自动引用计数
- 在工程中使用ARC只需要像往常一样写代码，只要不用retain,release,autorelease就好，这是ARC的基本原则
- 注意点
  - ARC时编译特性，不是运行特性
  - 它不是其他语言的gc，有本质区别
- 优点
  - 消除了内存管理的繁琐，避免内存泄漏，编译器优化
- ARC的判断原则
  - 只要还有一个强指针变量指向对象，对象就会保存在内存中
  - 默认所有指针变量都是强指针，被__strong修饰的指针
- 使用ARC的时候暂时忘记引用计数器，因为标准变了

```
int main(int argc, const char * argv[]) {
   @autoreleasepool {
        Person *p = [[Person alloc] init];
    } // 执行到这一行局部变量p释放
    // 由于没有强指针指向对象，所以对象也释放
    return 0;
}

int main(int argc, const char * argv[]) {
   @autoreleasepool {
        Person *p = [[Person alloc] init];
        p = nil; // 执行到这一行，由于没有强指针指向对象，所以对象被释放
    }
    return 0;
}
```

- 默认所有指针都是强指针，弱指针需要特别说明，**千万不要用弱指针保存新对象**
- 循环引用的时候，一方使用weak修饰

OC对象所有权修饰符	说明
__strong	对象默认修饰符，对象强引用，在对象超出作用域时失效。其实就相当于retain操作，超出作用域时执行release操作
__weak	弱引用，不持有对象，对象释放时会将对象置nil

__unsafe_unretained	弱引用，不持有对象，对象释放时不会将对象置nil
__autoreleasing	自动释放，由自动释放池管理对象

Block的循环引用问题

由于block会对block中的对象进行持有操作,就相当于持有了其中的对象，而如果此时block中的对象又持有了该block，则会造成循环引用。

https://www.jianshu.com/p/492be28d63c4