内存管理2

显示内存释放:C,C++

基于引用计数:OC

垃圾回收机制:Java,C#

自己生成的对象，自己持有 (alloc, allocWithZone:, copy, copyWithZone:, mutableCopy, mutableCopyWithZone:)

• 非自己生成的对象，自己也可以持有 (retain)

• 不再需要自己持有对象时，必须释放它 (release, autorelease)

• 非自己持有的对象无法释放

Run Loop 会在每次 loop 到尾部时销毁 Autorelease Pool

MRC : 忘记释放，内存泄露; 提前释放，悬停指针

ARC所做的只不不过是在代码编译时为你自动在合适的位置插入retain/release或autorelease，就如同之前MRC时你所做的那样

ARC

\_\_strong :

\_\_weak :不持有对象,当对象没有强引用的时候自动置为nil;

\_\_unsafe\_unretained :不持有对象,对象没有强引用的时候不会置为nil;

不能用强引用的实例变量;在C结构体里使用到Object-C对象;

\_\_autoreleasing :表明传引用的参数(id\*)在返回时是autorelease; 效果同MRC下调用autorelease方法

\_\_bridge 只是声明类型转变，但是不做内存管理规则的转变。

\_\_bridge\_retained 表示将指针类型转变的同时，将内存管理的责任由原来的Objective-C 交给Core Foundation 来处理，也就是，将 ARC 转变为 MRC。

\_\_bridge\_transfer 表示将管理的责任由 Core Foundation 转交给 Objective-C，即将管理方式由 MRC 转变为 ARC。

**任何继承了NSObject类的对象都需要内存管理,基本数据类型不需要(int,float,bool)**

内存分为堆和栈

栈:放局部产生的变量和对象的指针变量; int a;int b;

堆:放动态存储的对象变量 [[person alloc] init]

栈空间自动回收;堆空间手动释放;

当释放后的对象再次释放或再次进行相关操作就会报野指针错误;

指向僵尸对象的指针就叫野指针;

oc不会报空指针错误;

oc内存管理有加必有减,加减必相等;

1.你想使用（占用）某个对象，就应该让对象的计数器+1（让对象做一次retain操作）

2.你不想再使用（占用）某个对象，就应该让对象的计数器-1（让对象做一次release）

3.谁retain，谁release

4.谁alloc，谁release

5,内存管理对象要付责任,MRC内存管理相当严格;

//\*\*\*\*

1.set方法内存管理相关的参数

\* retain : release旧值，retain新值（适用于OC对象类型）

\* assign : 直接赋值（默认，适用于非OC对象类型）

\* copy : release旧值，copy新值

2.是否要生成set方法

\* readwrite : 同时生成setter和getter的声明、实现(默认)

\* readonly : 只会生成getter的声明、实现

3.多线程管理

\* nonatomic : 性能高 (一般就用这个)

\* atomic : 性能低（默认）

4.setter和getter方法的名称

\* setter : 决定了set方法的名称，一定要有个冒号 :

\* getter : 决定了get方法的名称(一般用在BOOL类型)

**循环引用**

循环引用:循环引用就是你引用我,我引用你;

Person类中import引用Card类,Card类中import引用Person类;这样就会相互引用;

[一般在.h文件中@class](mailto:一般在.h文件中@class) Person ,在.m文件中#import“”;

其中有一个对象的引用计数没有减到0,无法调用dealloc方法是导致循环引用的根本原因;

autorelsease不会增加引用;

自动释放池是以栈的形式存储对象;

autorelsease延迟释放对象,不能精确释放对象;

@autorelsease

{ //构建池子

}//释放对象

1.autorelease的基本用法

1> 会将对象放到一个自动释放池中

2> 当自动释放池被销毁时，会对池子里面的所有对象做一次release操作

3> 会返回对象本身

4> 调用完autorelease方法后，对象的计数器不变

2.autorelease的好处

1> 不用再关心对象释放的时间

2> 不用再关心什么时候调用release

3.autorelease的使用注意

1> 占用内存较大的对象不要随便使用autorelease

2> 占用内存较小的对象使用autorelease，没有太大影响

4.错误写法

1> alloc之后调用了autorelease，又调用release

@autoreleasepool

{

// 1

Person \*p = [[[Person alloc] init] autorelease];

// 0

[p release];

}

2> 连续调用多次autorelease

@autoreleasepool

{

Person \*p = [[[[Person alloc] init] autorelease] autorelease];

}

5.自动释放池

1> 在iOS程序运行过程中，会创建无数个池子。这些池子都是以栈结构存在（先进后出）

2> 当一个对象调用autorelease方法时，会将这个对象放到栈顶的释放池

6.自动释放池的创建方式

1> iOS 5.0前

NSAutoreleasePool \*pool = [[NSAutoreleasePool alloc] init];

[pool release]; // [pool drain];

2> iOS 5.0 开始

@autoreleasepool

{

}