**内存管理3+P**

/\*

1.方法的基本使用

1> retain ：计数器+1，会返回对象本身

2> release ：计数器-1，没有返回值

3> retainCount ：获取当前的计数器

4> dealloc

\* 当一个对象要被回收的时候，就会调用

\* 一定要调用[super dealloc]，这句调用要放在最后面

2.概念

1> 僵尸对象 ：所占用内存已经被回收的对象，僵尸对象不能再使用

2> 野指针 ：指向僵尸对象（不可用内存）的指针，给野指针发送消息会报错（EXC\_BAD\_ACCESS）

3> 空指针 ：没有指向任何东西的指针（存储的东西是nil、NULL、0），给空指针发送消息不会报错

\*/

/\*

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

\*/

/\*

内存管理代码规范：

1.只要调用了alloc，必须有release（autorelease）

对象不是通过alloc产生的，就不需要release

2.set方法的代码规范

1> 基本数据类型：直接复制

- (void)setAge:(int)age

{

\_age = age;

}

2> OC对象类型

- (void)setCar:(Car \*)car

{

// 1.先判断是不是新传进来对象

if ( car != \_car )

{

// 2.对旧对象做一次release

[\_car release];

// 3.对新对象做一次retain

\_car = [car retain];

}

}

3.dealloc方法的代码规范

1> 一定要[super dealloc]，而且放到最后面

2> 对self（当前）所拥有的其他对象做一次release

- (void)dealloc

{

[\_car release];

[super dealloc];

}

\*/

// 速度为100的车子：

// 速度为200的车子：p1->\_car

//- (void)setBook:(Book \*)book

//{

// if (\_book != book)

// {

// [\_book release];

//

// \_book = [book retainonatomic, n];

// }

//}

//如果旧书不等于新书,那应该把旧书扔掉,再使用(retain)新书; -->内存管理的set方法 ;

//release方法一般放在方法最后

//操控内存管理的方法,主要存在于set方法,和dealloc方法;

/\*

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)

\*/

/\*

1.@class的作用：仅仅告诉编译器，某个名称是一个类

@class Person; // 仅仅告诉编译器，Person是一个类

2.开发中引用一个类的规范

1> 在.h文件中用@class来声明类

2> 在.m文件中用#import来包含类的所有东西

3.两端循环引用解决方案

1> 一端用retain

2> 一端用assign

\*/

/\*

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

{

}

\*/

/\*

1.系统自带的方法里面没有包含alloc、new、copy，说明返回的对象都是autorelease的

2.开发中经常会提供一些类方法，快速创建一个已经autorelease过的对象

1> 创建对象时不要直接用类名，一般用self

+ (id)person

{

return [[[self alloc] init] autorelease];

}

\*/

/\*

ARC的判断准则：只要没有强指针指向对象，就会释放对象

1.ARC特点

1> 不允许调用release、retain、retainCount

2> 允许重写dealloc，但是不允许调用[super dealloc]

3> @property的参数

\* strong ：成员变量是强指针（适用于OC对象类型）

\* weak ：成员变量是弱指针（适用于OC对象类型）

\* assign : 适用于非OC对象类型

4> 以前的retain改为用strong

指针分2种：

1> 强指针：默认情况下，所有的指针都是强指针 \_\_strong

2> 弱指针：\_\_weak

\*/

/\*

当两端循环引用的时候，解决方案：

1> ARC

1端用strong，另1端用weak

2> 非ARC

1端用retain，另1端用assign

\*/

/\*

block要掌握的东西

1> 如何定义block变量

int (^sumBlock)(int, int);

void (^myBlock)();

2> 如何利用block封装代码

^(int a, int b) {

return a - b;

};

^() {

NSLog(@"----------");

};

^ {

NSLog(@"----------");

};

3> block访问外面变量

\* block内部可以访问外面的变量

\* 默认情况下，block内部不能修改外面的局部变量

\* 给局部变量加上\_\_block关键字，这个局部变量就可以在block内部修改

4>　利用typedef定义block类型

typedef int (^MyBlock)(int, int);

// 以后就可以利用MyBlock这种类型来定义block变量

MyBlock block;

MyBlock b1, b2;

b1 = ^(int a, int b) {

return a - b;

};

MyBlock b3 = ^(int a, int b) {

return a - b;

};

\*/