16-【掌握】autorelease基本使用

本小节知识点：

1、【了解】autorelease是什么？
2、【理解】为什么会有autorelease？
3、【理解】autorelease的原理？
4、【理解】autorelease什么时候被释放？

1.原理

autorelease

是一种支持引用计数的内存管理方式

它可以暂时的保存某个对象（object），然后在内存池自己的排干（drain）的时候对其中的每个对象发送release消息

注意，这里只是发送release消息，如果当时的引用计数（reference-counted）依然不为0，则该对象依然不会被释放。可以用该方法来保存某个对象，也要注意保存之后要释放该对象。

自动释放池 :特殊的栈结构
特点: 对象可以加入到自动释放池中
自动释放池结束的时候,会给池中的对象发送一条 release消息

自动释放池的使用:

1) 创建自动释放池

```
@autoreleasepool {

}
```

2) 加入自动释放池
在自动释放池中

```
[对象 autorelease];
```

```
int main(int argc, const char * argv[]) {
    //1 创建自动释放池
    Person *p = [Person new];  // p  1
    @autoreleasepool {//自动释放池开始

        [p run];
        NSLog(@"%lu",p.retainCount); // 1

        // [p autorelease] 把对象p加入到自动释放池中
        // 注意:加入到自动释放池中以后, 引用计数不会变
        [p autorelease];  //加入自动释放池,
        NSLog(@"%lu",p.retainCount); // 1

        [p run];

    }//自动释放池结束
    return 0;
```

自动释放池只会释放一次,如果在释放池中retain,是无法释放完毕的

- 问题:
  1，autorelease的原理是什么？
  2，使用autorelease的好处

2.好处

使用autorelease的好处

（1）不需要再关心对象释放的时间

（2）不需要再关心什么时候调用release

1、自动释放池及autorelease介绍

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（先进后出）存在的。

（2）当一个对象调用autorelease时，会将这个对象放到位于栈顶的释放池中

自动释放池的创建方式

（1）iOS 5.0以前的创建方式

```
NSAutoreleasePool *pool=[[NSAutoreleasePool alloc] init];
.....
[pool  release];//[pool drain];用于mac
```

（2）iOS5.0以后

```
@autoreleasepool
{//开始代表创建自动释放池
.....
}//结束代表销毁自动释放池
```

本小节知识点：

1、【掌握】autorelease使用注意

2、【理解】autorelease错误用法

1、autorelease使用注意

1）并不是放到自动释放池代码中，都会自动加入到自动释放池

```
//1 自动释放池
Person *p = [Person new];
@autoreleasepool {

    //autorelease的使用注意:
    // 1)并不是所有的放到自动释放池中的代码,产生的对象就会自动释放
    //    如果需要释放,必须加入到自动释放池
//    Person *p = [[Person new] autorelease];

    //我们只需要在自动释放池代码块中调用autorelease 就可以把对象
    //加入到自动释放池
    [p autorelease];

}

//2) 如果对象调用了autorelease 但是,调用autorelease的时候,没有在任
    何一个自动释放池中,此时该对象也不会 被加入到自动释放池
//  Person *p = [[Person new] autorelease];

return 0;
```

第二种

2.自动释放池的嵌套使用

```
//autoreleasepool的嵌套
//自动释放池的栈结构(数据结构),和内存的栈区是不一样的
// 对象存在 位于栈顶位置的自动释放池中
@autoreleasepool {

    @autoreleasepool {

        @autoreleasepool {

            [p autorelease];

        }// [p release];  //1

        [p autorelease];

    }//

}
```

每次释放池都release了一次,不允许再一个池子里面使用两次

3）自动释放池中不适宜放占用内存比较大的对象

1）尽量避免对大内存使用该方法，对于这种延迟释放机制，还是尽量少用

2）不要把大量循环操作放到同一个 @autoreleasepool 之间，这样会造成内存峰值的上升

2、autorelease错误用法

（1）连续调用多次autorelease，释放池销毁时执行两次release(-1吗？)

（2）Alloc之后调用了autorelease，之后又调用了release。

```
// 对象存在 位于栈顶位置的自
@autoreleasepool {

    [p autorelease];

}//   [p release];    1-

[p release]; //guo'wu
```

1.错误 2.错 3

应用场景

- 问题
- 1，使用instancetype的好处

```
//id 和 instancetype
// instancetype 可以智能帮我们判断赋值赋值的指针变量的类型和方法
    的返回值类型是否一致
```

本小节知识点：
1、【掌握】autorelease的应用场景
2、【掌握】完善快速创建对象的方法

1、autorelease的应用场景

经常用来在类方法中快速创建一个对象

快速创建对象并释放?需要再类方法实现.

```
+(instancetype)studentWithAge:(int)age{

    return [[[self alloc] initWithAge:age] autorelease];

}
```

直接调用类方法

2、完善快速创建对象的方法

```
Student *stu = [[Student alloc] initWithAge:18];
NSLog(@"stu.age = %d",stu.age);
```

4-【理解】应用：创建一个学生类初始化年龄

思考&实现1：
创建一个学生类Student，通过重写构造方法实现创建学生对象的时候默认的年龄值指定的年龄

自定义一个方法,初始化子类自己的年龄

```
//自定义一个构造方法
-(instancetype)initWithAge:(int)age{

    //1 先初始化父类的,并且判断是否初始化成功
    if (self = [super init]) {
    //2 初始化子类的
        _age = age;
     }
    //3 返回self
    return self;
}
```

```
Student *stu = [[Student alloc] initWithAge:18];
NSLog(@"stu.age = %d",stu.age);
```

上面还要再release下面最终版

```
}
+(instancetype) studentWithAge: (int) age {

        return [[[self alloc] initWithAge: age
            ] autorelease];

}
```

```
// 2) 类方法有参数,传递一个年龄
Student *stu2 = [Student studentWithAge:38];
NSLog(@"stu2.age = %d",stu2.age);
```