第二十九条：理解引用计数

理解引用计数原理

在引用计数架构下，每个对象都有一个引用计数器，用来表示到目前还有多少个事物令对象继续存活下去，在Objectice-C中叫引用计数或者保留计数。NSObject协议中三种方法用于操作此：

- retain：递增保留计数；
- release：递减保留计数；
- autorelease：待稍后清理自动释放池（autoreleasepool）时再递减引用计数。

在Objective-C中调用allocation方法返回的对象由调用者所拥有。

属性存取方法中的内存管理

在访问对象的属性时，会用到相关实例变量的取值方法和设值方法，若属性为strong强引用，则在设置值时会保留属性值。比如strong类型的name属性的设值函数会变成这个样子：

```
-(void)setName:(NSString *)name{
    [name retain]
    [_name release]
    _name = name;
}
```

所以这里是先保留新值，再释放旧值，最后更新实例变量。

自动释放池

在Objective-C中自动释放池是一种重要的机制，**一般的释放操作会在执行之后立马让引用计数器减一，但是自动释放池却可以延迟此过程，通常会在下一个事件循环时递减**，也有可能执行地早点。

循环引用retain cycle

循环引用就是多个对象相互引用对方，形成“信息孤岛”，因为他们的计数器最后都会保留1，所以造成内存泄漏。通常，对于此类问题应该用弱引用weak来解决。

要点：

1. 引用计数机制通过可以递增和递减的计数器来管理内存。对象创建完成之后，其保留计数至少为1。若保留计数为正，对象继续存活；若保留计数为0，对象被销毁；
2. 在对象生命周期中，其余对象通过引用来保留或者释放对象。保留和释放会分别递增和递减保留计数。

第三十条：用ARC简化引用计数

Clang编译器自带一个“静态分析器”，用于指明程序里引用计数出问题的地方，并自动加上应该有的内存管理语句，这是最重要的。由于ARC会自动执行retain，release，autorelease等操作，所以直接在ARC下面调用这些内存管理方法是非法的，具体来说有：

- retain
- release
- autorelease
- dealloc

ARC在调用这些方法时，并不是通过Objectice-C的消息派发机制，而是通过底层的C语言执行的，因为这样做性能会更好。

使用ARC是必须遵循的命名规则

将内存管理语义在方法名中表示出来早已成为Objective-C的惯例，而ARC则将之确立为硬性规定。若方法名以下面词语开头，则其返回的对象归调用者所有：

- alloc
- new
- copy
- mutableCopy

归调用者所有的意思是：调用上述四种方法的代码要负责释放掉方法所返回的对象。也就是说，这些对象的保留计数为正数，调用了这四种方法的那段代码要将其中一次保留retain操作抵消掉release。若方法名不以这四个词语开头，则表示所返回的对象不归调用者所有。

除了自动调用retain和release之外，使用ARC还有很多好处，它可以执行一些手工操作很难甚至无法完成的操作。例如，在编译器，ARC会把能够互相抵消的retain、release和autorelease操作简约。如果发现在同一个对象上执行了多次retain和release，那么ARC有时会成对的移除这两个操作。

变量的内存管理语义

ARC也会处理局部变量和实例变量的内存管理。默认情况下，**每个变量都是指向对象的强引用**。

ARC会用一种安全的方式来设置：**先保留新值再释放旧值，最后设置实例变量**。

在应用程序中，可以用下列修饰符来改变局部变量和实例变量的语义：

- __strong：默认语义，保留此值；
- __unsafe_unretained：不保留此值，这么做可能不安全，因为等到再次使用变量是，其对象可能已经回收了；
- __weak：不保留此值，但是变量可以安全使用，因为如果系统吧这个对象回收了，那么变量会自动清空。
- __autoreleasing：把对象"按引用传递"给方法时，使用这个特殊的修饰符，此值在方法返回时自动释放。

我们经常会给局部变量加上修饰符__weak，用以消除由block所引入的循环引用问题。

ARC如何清理实例变量

在MRC下需要在dealloc方法中释放对象。在ARC下就不需要编写这种dealloc方法了，因为ARC会借用Objective-C++的一项特性生成清理例程。回收Objective-C++对象时，待回收的对象会调用所有C++对象的析构函数（destructor）。编译器如果发现某个对象里面含有C++对象，就会生成.cxx_destrut的方法。

要点：

1. 有ARC之后，程序员就不需要担心内存管理问题了。使用ARC来编程，可以省去类中的许多"样板代码"；
2. ARC管理对象生命期的办法基本上是：在合适的地方插入retain及release操作。在ARC环境下，变量的内存管理语义可以通过修饰符指明，而原来需要手动执行retain和release操作；
3. 由方法所返回的对象，其内存管理语义总是通过方法名来体现。ARC将此确定为开发者必须遵守的规则；
4. ARC值负责管理Objective-C对象的内存，尤其要注意：CoreFoundation对象不归ARC管理，开发者必须适时调用CFRetain/CFRelease。

第三十一条：在dealloc方法中只释放引用并解除监听

虽然dealloc方法中应该释放引用，但是开销较大或者系统内的稀缺资源不在此列。比如文件描述符，套接字或者大内存块。比如用于连接服务器的套接字，应该有如下方法：

```
-(void)open:(NSString *)address;
-(void)close;
```

在close方法中清理资源的另一个原因是：系统并不能保证每个对象的dealloc方法执行。

要点：

1. 在dealloc方法中应该做的事是释放指向其他对象的引用，并且还要取消KVO和监听；
2. 如果对象持有文件描述符等系统资源，那么应该专门编写一种方法释放此种资源。这样的类和使用者约定：用完资源必须调用close方法；
3. 执行异步任务的方法不应该在dealloc方法中；只能在正常状态下执行那些方法，因为此时对象已经处于正在回收的状态了。

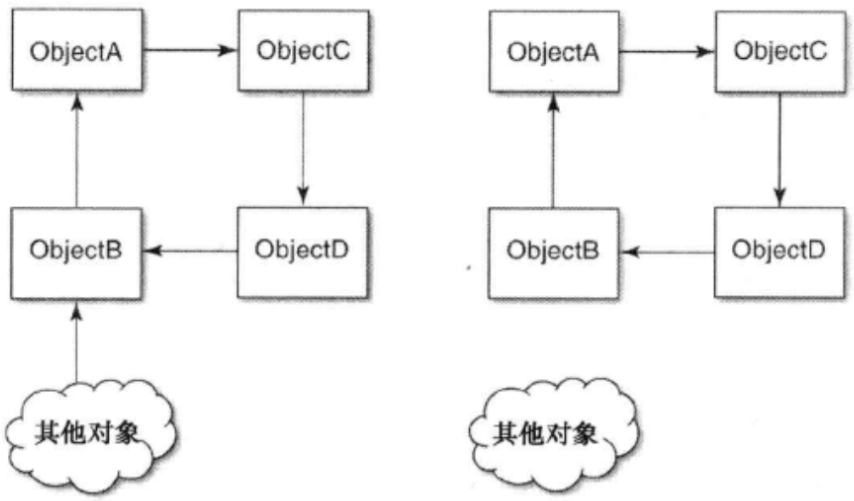
第三十二条：编写“异常安全代码”时留意内存管理问题

要点：

1. 捕获异常时，一定要注意将try块内创立的对象清理干净；
2. 在默认情况下，ARC不生成安全处理异常所需的清理代码。用-jobjc-arc-exceptions开启标志位后可生成这种代码，不过会导致应用程序变大，降低运行效率。

第三十三条：以弱引用避免循环引用

循环引用问题：



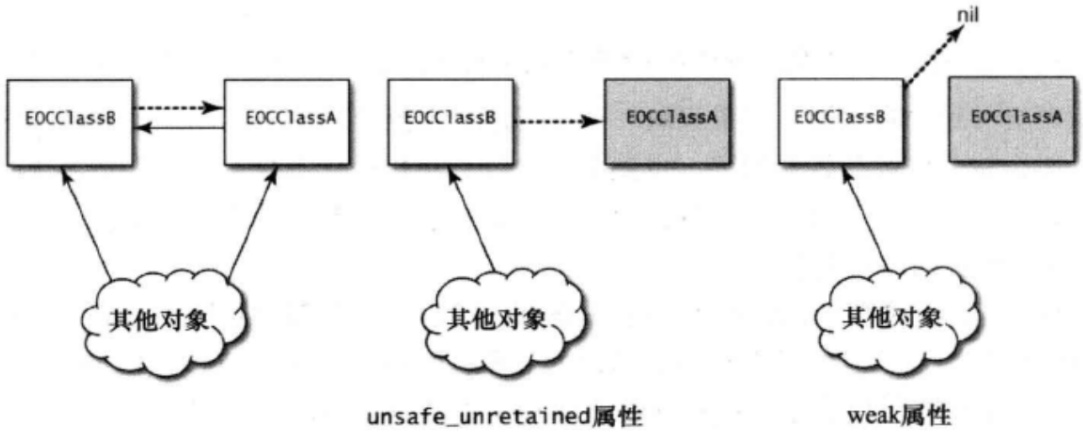


图 5-6　unsafe_unretained 与 weak 属性，在其所指的对象回收以后表现出来的行为不同

一般来说，如果不拥有某个对象，那就不要保留它。这条规则对collection例外，collection虽然并不直接拥有其内容，但是它要代表所属的那个对象来保留这些元素。有时，对象中的引用会指向另外一个并不归自己所拥有的对象，比如delegate模式就是这样。

要点：

1. 将某些属性设置成weak，可以避免出现循环引用；
2. weak引用可以自动清空，也可以不自动清空，自动清空（autoniling）是随着ARC而引入的新特性，由运行时系统来实现，在具备自动清空的弱引用上，可以随意读取数据，因为这种引用不会指向已经回收过的对象。

第三十四条：用自动释放池块降低内存峰值

在Objective-C的引用计数中，有一种特性叫做“自动释放池”。释放有对象的方法：一种是release方法，另一种是autorelease方法，将其加入“自动释放池”中。自动释放池用于那些在稍后释放的对象。

看下面这段代码：

```
for (i = 0;i < 10000;i++) {
    [self doSomeThingWithInt:i];
}
```

在这个for循环中，创建了大量的临时变量。但是这些这些对象在调用完方法之后就不在使用了，它们依然处于存活状态，因为目前还在入口程序的自动释放池中，所以在等待系统稍后将其释放，也就是说只能在for循环完成时清理所有的临时变量。这样一来，在执行for循环时内存占用量会急剧上升，等到结束时，急剧下降。

其实这些临时变量是可以及时回收的，我们在for循环中加上**@autoreleasepool{[self doSomeThingWithInt:i];}**。这样在循环时我们创建的这个自动释放池就会及时回收它们。这样就可以降低内存峰值（应用程序在特定时间段的最大内存使用量）。

```
for (i = 0;i < 10000;i++) {
    @autoreleasepool {
        [self doSomeThingWithInt:i];
    }
}
```

要点：

1. 自动释放池排布在栈中，对象收到autorelease消息后，系统将其放入到最顶端的池中；
2. 合理应用自动释放池，降低应用程序内存峰值；
3. @autoreleasepool这种新式写法能创建出更为轻便的自动释放池。

第三十五条：使用“僵尸对象”调试内存管理问题

为了调试时防止在对象被回收之后发送消息而造成不安全问题，Cocoa提供了“僵尸对象”（Zombie object）这个非常方便的功能，启用这项调试功能之后，运行时系统会把所有的已经回收的实例转换成特殊的“僵尸对象”，而不会真正回收它们，这种对象所在的核心内存无法重用，因此不可能遭到覆写。

僵尸对象的工作原理：它的实现代码深植于Objectice-C的运行时库。Foundation框架以及CoreFoundation框架中。系统在即将回收对象时，如果通过环境变量启用了僵尸对象，那么还将执行一个附加步骤。这一步就是把原对象转换成僵尸对象，而不彻底回收。

要点：

1. 系统在回收对象时，可以不将其真的回收，而是把它转换成僵尸对象。通过环境变量NSZombieEnabled开启此功能；
2. 系统会修改对象的ISA指针，令其指向特殊的僵尸类，从而使该对象变成真正的僵尸对象。僵尸类能够响应所有的方法，响应方式为：**打印一条包含消息内容及其接收者的消息，然后终止应用程序**。

第三十六条 ： 不要使用retainCount

要点：

1. 对象的保留计数看似有用，实则不然，因为任何时间点上的“绝对保留计数”都无法反映对象生命期的全貌；

2. 引入ARC之后，retainCount方法就正式废止了，在ARC下调试会报错。