# 文档前言

## 基本描述

俗话说的好：“好记性，不如烂笔头”。说明的一个道理就是纪录的重要性，哪怕是一个记忆天才，纪录在其他地方，可以随时看到也是非常重要的。何况我是如此一个平凡的人啊！谦虚学习，踏踏实实，不能好高骛远，天道酬勤才是王道。

本文档旨在纪录开发过程中遇到的一些重要点、难点，也能帮助我纪录我的成长过程。

少波，好好纪录，这是很重要的一个事情，不要觉得我写了一些知道的东西，没什么价值，磨刀不误砍柴工。

## 内容概括

本文包括如下内容：iOS开发、设计模式、数据结构、java服务器开发、前段开发等，若有新技术的添加会及时的添加到文档中。

IOS开发部分

# Ios代码书写规范

## 枚举的写法：

# 网络部分

## 实现离线断点下载功能（2016年4月19日星期二）

### 基本描述：

实现大文件的离线重新下载，在程序运行过程可以暂停、继续以及在程序app下一次启动后，可以继续下载，上次下载的文件进行保存，下次从上次下载完成的部分继续下载。

基本的实际案例：百度音乐下载、喜马拉雅音乐下载、百度云文件的下载。

其实本人认为，网络部分断点续传算是难度和复杂度都比较高的部分。

### 解决方案

#### 解决方案理论

用户发送下载文件请求（如下载音乐、大文件等），当用户首次发送下载文件，将请求头部设置为请求的文件二进制位置为0，在下载过程中不断将下载的二进制文件存储在本地，并且记录下载量，一旦用户暂停下载，app记录下最后下载完成的数量以及下载请求的位置，从多少字节开始下载。服务器端则根据客户端的请求，从head区域的文件请求位置，开始传输二进制数据，发送给客户端，在完成总的文件下载后，客户端取消任何关于本文件的下载信息。

当前，一般的网络处理，都会用到AFNetworking 网络请求库，但是此框架对实现离线形式（重启app，从上次的下载点继续下载）的断点续传支持不好。因此我此处使用到的是apple系统api原生的网络请求方式。

#### 解决实践

1. 用到NSURLSession ->session，调用含有代理设置的初始化方法:

+ (NSURLSession \*)sessionWithConfiguration:(NSURLSessionConfiguration \*)configuration delegate:(nullable id <NSURLSessionDelegate>)delegate delegateQueue:(nullable NSOperationQueue \*)queue;

2. 使用NSMutableRequest->request，设置头部文件为：

//设置表头信息

NSString \*rang = [NSString stringWithFormat:”@bytes=%zd-”,currrentSize];

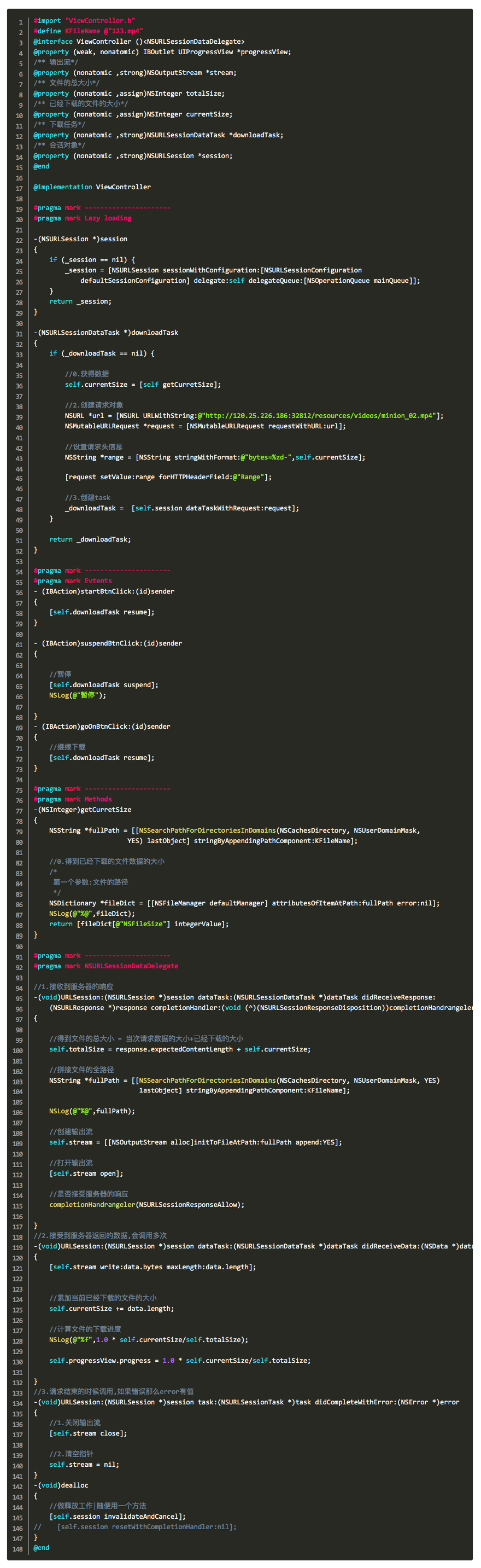
[request setValue:rang forHTTPHeanderField:”@Range”];

3.调用创建task方法

//创建task

NSURLSessionDataTask \*task = [session dataTaskWithRequest:request];

下图是参考别人写的，我觉的很好，直接贴上来，如果觉的图片不清楚，可以通过放大文档的形式查看，如下图：



# 多线程

## NSThread

是三种方法里面相对轻量级的，但需要管理线程的生命周期、同步、加锁问题，这会导致一定的性能开销

### 优缺点

1.优点：NSThread比其他两种多线程方案较轻量级，更直观地控制线程对象

2.缺点：需要自己管理线程的生命周期，线程同步。线程同步对数据的加锁会有一定的系统开销

## Cocoa Operations(NSOperationQueue)

是基于OC实现的，NSOperation以面向对象的方式封装了需要执行的操作，不必关心线程管理、同步等问题。NSOperation是一个抽象基类，iOS提供了两种默认实现：NSInvocationOperation和NSBlockOperation，当然也可以自定义NSOperation

### 注意点：

1. 运行中的任务不能够取消，只能够取消还没有执行的任务。
2. 分为两个队列，主线程队列和新线程队列。
3. 在NSOperationQueue中, 可以指定各个NSOperation之间的依赖关系.
4. 主队列只是串行执行
5. operation的executing和finished状态量需要用willChangeValueForKey/didChangeValueForKey来触发KVO消息.

### 与gcd对比

NSOperationQueue 是在GCD基础上实现的，只不过是GCD更高一层的抽象。

GCD 只支持FIFO 的队列， 而NSOperationQueue可以调整队列的执行顺序。（通过调整权重）

NSOperationQueue可以在Operation间设置依赖关系，而GCD不可以。 如果一个Operation依赖另一个Operation所产生的数据的化，你可以设置一个Operation依赖于另一个Operation来实现， NSOperationQueue可以根据依赖关系，可以以正确的顺序执行Queue中的Operation。

NSOperationQueue支持KVO。 这就意味着你可以观察任务的状态属性。

以上几点并不是说在任何处理多任务时一定要选择NSOperationQueue， 因为NSOperationQueue在执行速度上会比GCD慢。

## GCD（grand gentral dispatch）

简称GCD，iOS4才开始支持)：提供了一些新特性、运行库来支持多核并行编程，它的关注点更高：如何在多个cpu上提升效率

## 同步锁（NSLock）

## @ synchronized

使用方法；

在防止多个线程访问的对象和代码处使用如下：

@synchronized (\_elements) {

[\_elements addObject:element];

}

”synchronized block” 与 [\_lock lock] 和 [\_lock unlock] 效果相同。你可以把它当成是锁住 self，仿佛 self 就是个 NSLock。锁在左括号 { 后面的任何代码运行之前被获取到，在右括号 } 后面的任何代码运行之前被释放掉。这爽就爽在妈妈再也不用担心我忘记调用 unlock 了！

其实上面的\_elements可以设置为当前的类对象，self，关键是通过@synchronized关键字会将过好内的代码进行加锁，等代码执行完成后才允许其他线程访问这些代码。

# NSRunLoop(较难)

# Block

## Block简介

Block是一种比较特殊的数据类型。它可以保存一段代码，在合适的时候取出来调用。

## Block的定义

Block的定义格式

返回值类型(^block变量名)(形参列表) = ^(形参列表) {

};

调用Block保存的代码

block变量名(实参);

默认情况下,Block内部不能修改外面的局部变量

Block内部可以修改使用\_\_block修饰的局部变量

Block的模式

1.无参数无返回值的Block

2.有参数无返回值的Block

3.有参数有返回值的Block

## Block的修饰

ARC情况下

1.如果用copy修饰Block，该Block就会存储在堆空间。则会对Block的内部对象进行强引用，导致循环引用。内存无法释放。

解决方法：

新建一个指针(\_\_weak typeof(Target) weakTarget = Target )指向Block代码块里的对象，然后用weakTarget进行操作。就可以解决循环引用问题。

2.如果用weak修饰Block，该Block就会存放在栈空间。不会出现循环引用问题。

MRC情况下

用copy修饰后，如果要在Block内部使用对象，则需要进行(\_\_block typeof(Target) blockTarget = Target )处理。在Block里面用blockTarget进行操作。

## Block简单用法举例

### 无参数无返回值的Block

/\*\*

\* 无参数无返回值的Block

\*/

-(void)func1{

/\*\*

\* void ：就是无返回值

\* emptyBlock：就是该block的名字

\* ()：这里相当于放参数。由于这里是无参数，所以就什么都不写

\*/

void (^emptyBlock)() = ^(){

NSLog(@"无参数,无返回值的Block");

};

emptyBlock();

}

### 有参数无返回值的Block

/\*\*

\* 调用这个block进行两个参数相加

\*

\* @param int 参数A

\* @param int 参数B

\*

\* @return 无返回值

\*/

void (^sumBlock)(int ,int ) = ^(int a,int b){

NSLog(@"%d + %d = %d",a,b,a+b);

};

/\*\*

\* 调用这个sumBlock的Block，得到的结果是20

\*/

sumBlock(10,10);

### 有参数有返回值的Block

/\*\*

\* 有参数有返回值

\*

\* @param NSString 字符串1

\* @param NSString 字符串2

\*

\* @return 返回拼接好的字符串3

\*/

NSString\* (^logBlock)(NSString \*,NSString \*) = ^(NSString \* str1,NSString \*str2){

return [NSString stringWithFormat:@"%@%@",str1,str2];

};

//调用logBlock,输出的是 我是Block

NSLog(@"%@", logBlock(@"我是",@"Block"));

### Block结合typedef使用

自己定义一个Block类型，用定义的类型去创建Block，更加简单便捷。  
这里举例一个Block回调修改上一下界面的背景颜色。  
ViewController1 控制器1，ViewController2 控制器2  
控制器1跳转到控制器2，然后在控制器2触发事件回调修改控制器1的背景颜色为红色。

ViewController2的实现

#import <UIKit/UIKit.h> @interface ViewController2 : UIViewController /\*\* \* 定义了一个changeColor的Block。这个changeColor必须带一个参数，这个参数的类型必须为id类型的 \* 无返回值 \* @param id \*/ typedef void(^changeColor)(id); /\*\* \* 用上面定义的changeColor声明一个Block,声明的这个Block必须遵守声明的要求。 \*/ @property (nonatomic, copy) changeColor backgroundColor; @end

-(void)touchesBegan:(NSSet<UITouch \*> \*)touches withEvent:(UIEvent \*)event{ //声明一个颜色 UIColor \*color = [UIColor redColor]; //用刚刚声明的那个Block去回调修改上一界面的背景色 self.backgroundColor(color); }

ViewController1的实现

-(void)touchesBegan:(NSSet<UITouch \*> \*)touches withEvent:(UIEvent \*)event{ ViewController2 \*vc =[[ViewController2 alloc]init]; // 回调修改颜色 vc.backgroundColor = ^(UIColor \*color){ self.view.backgroundColor = color; }; [self.navigationController pushViewController:vc animated:YES]; }

# 内存管理

## 检测内存泄漏

**Xcode结合Leaks检测内存泄露**

字数329 阅读503 评论1 喜欢18

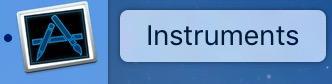
Collection/Bookmark/Share for width under 768px

**内存很重要！内存很重要！内存很重要！**

重要的事情要说三遍。虽然现在项目大多数是ARC模式，Xcode会帮我们管理内存，但并不意味着这样就不会存在内存泄露的现象。Xcode里面有一个工具Instruments，这个工具可以检测项目的性能。

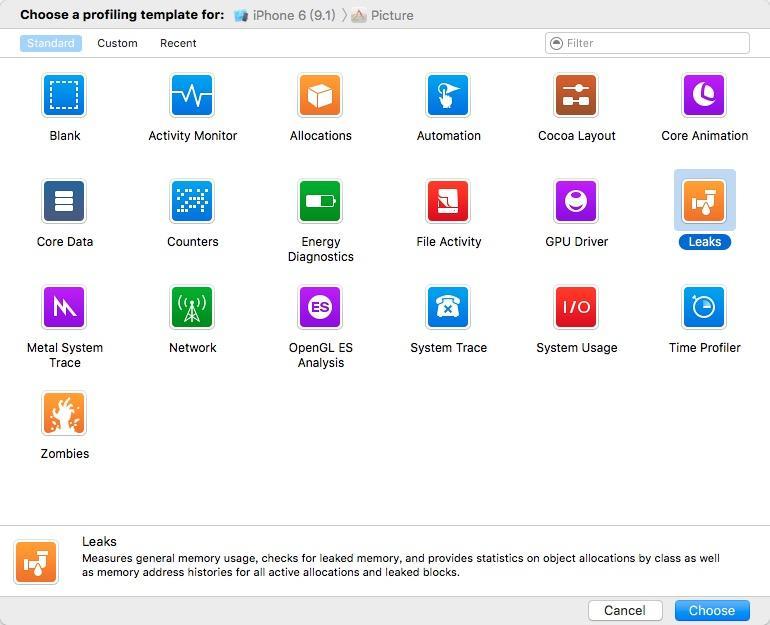
**Xcode结合Leaks检测内存泄露代码**

1.command+i 开启 Instruments工具



启动.png

2.选择Leaks，点choose



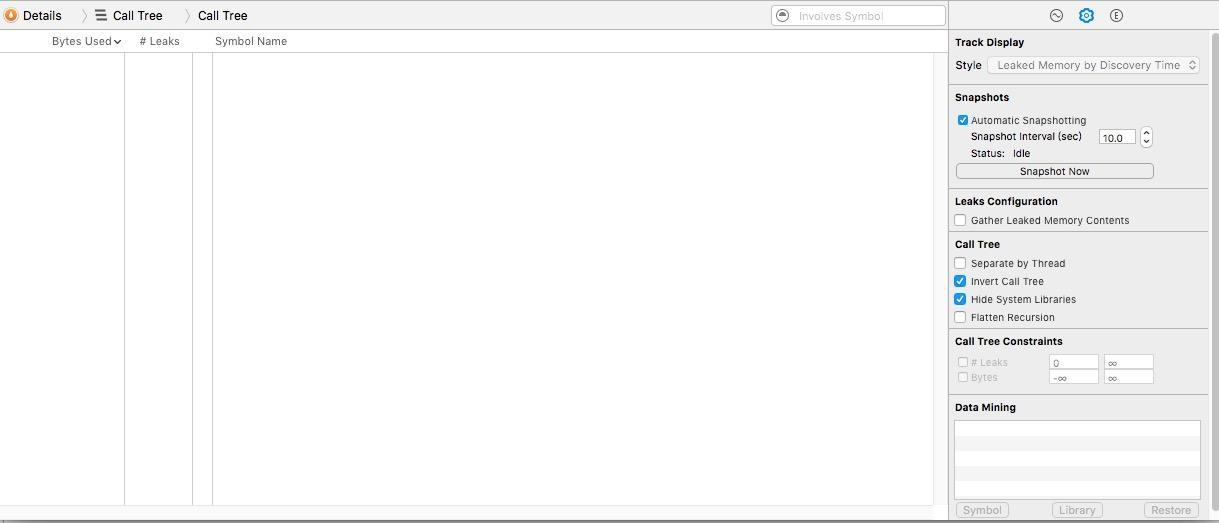
Leaks.png

3.选择Leaks，然后在Leaks那里选择**Call Tree**



xxxx.png

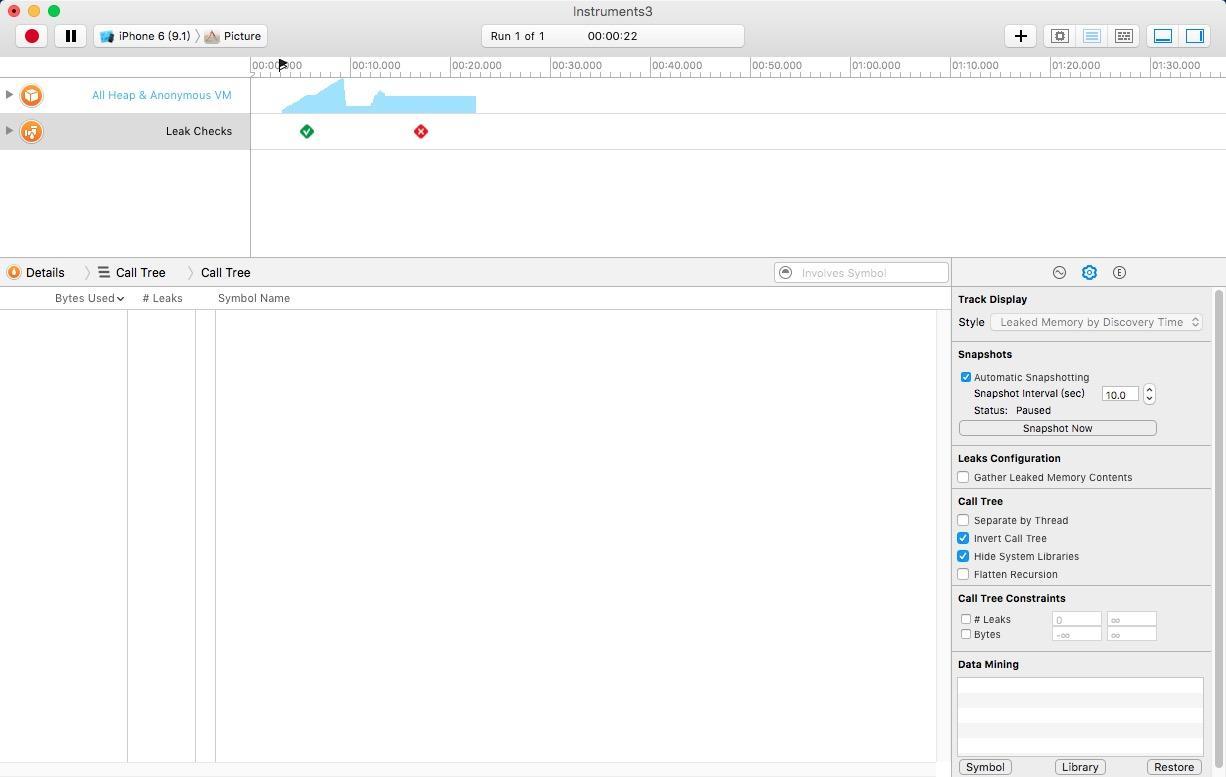
4.选择好Call Tree 之后选择设置，长得和齿轮一样



xxxxx.png

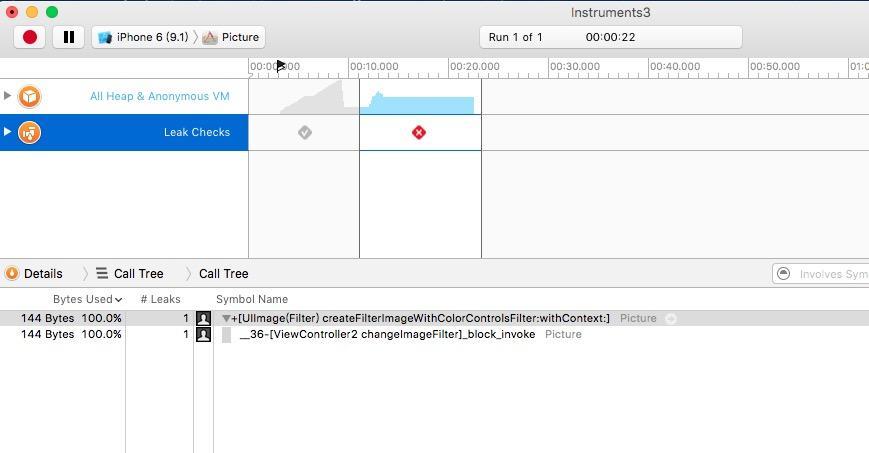
把Invert Call Tree 和 Hide system Libraries勾选

5.最后点左上角的红点Record，你就会发现模拟器启动了，然后随便你操作app。如果出现了红色的叉叉，就说明存在内存泄露。



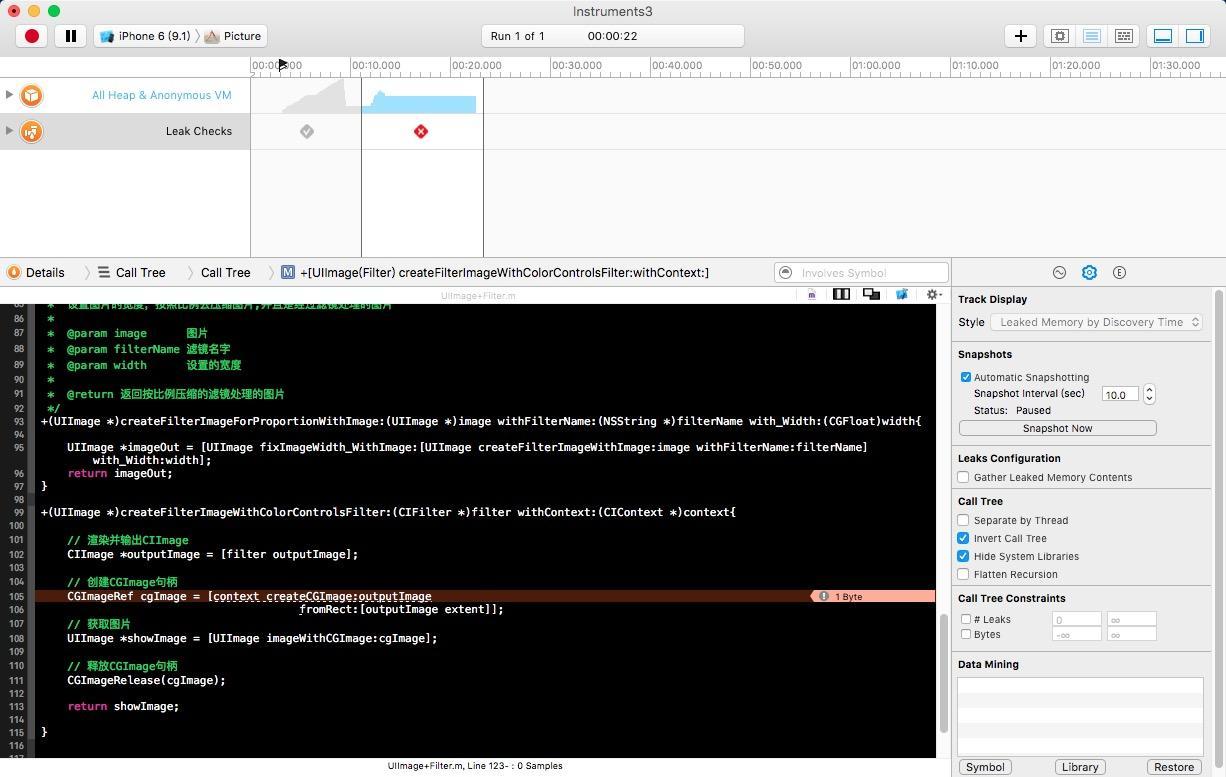
Bug.png

6.选中红色叉叉的区域,在Details里面就可以知道泄露的代码具体在哪一行。双击那段代码就可以去到泄露的位置。



findBug.png

7.找到泄露的代码，也可以一直点进去看。



Bug11.png

8.结束啦，找到泄露的地方，那就改去咯。

##### 内存管理

堆：需要手动释放。  
栈：系统自动管理。

##### 所谓内存管理,就是对内存进行管理,涉及的操作有:

分配内存 :比如创建一个对象,会增加内存占用  
清除内存 :比如销毁一个对象,能减小内存占用

##### 内存管理的管理范围

任何继承了NSObject的对象  
对其他非对象类型无效(int、char、float、double、struct、enum等)

##### 只有OC对象才需要进行内存管理的本质原因

OC对象存放于堆里面  
OC基础类型一般放在栈里面(栈内存会被系统自动回收)

##### 引用计数器的常见操作

给对象发送一条retain消息,可以使引用计数器值+1（retain方法返回对象本身）  
给对象发送一条release消息,可以使引用计数器值-1  
给对象发送retainCount消息,可以获得当前的引用计数器值

##### 需要注意的是: release并不代表销毁\回收对象,仅仅是计数器-1。

##### 空指针\野指针

僵尸对象  
已经被销毁的对象(不能再使用的对象)

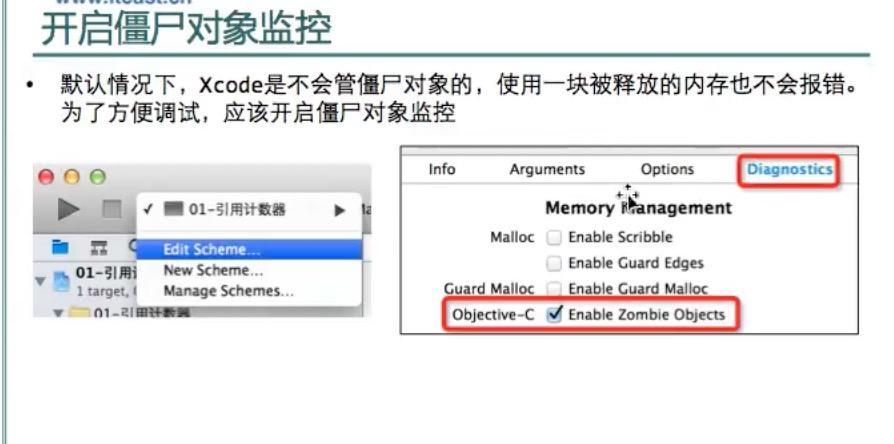
野指针  
指向僵尸对象(不可用内存)的指针  
给野指针发消息会报**EXC\_BAD\_ACCESS**错误

空指针  
没有指向存储空间的指针(里面存的是nil,也就是0)  
给空指针发消息是没有任何反应的。  
为了避免野指针错误的常见办法  
在对象被销毁之后,将指向对象的指针变为空指针

##### delloc方法注意细节

**当一个对象的引用计数器值为0时**  
这个对象即将被销毁，其占用的内存被系统回收  
系统会自动给对象发送一条dealloc消息  
(因此,从dealloc方法有没有被调用,就可以判断出对象是否被销毁)  
**dealloc方法的重写**  
一般会重写dealloc方法,在这里释放相关资源,dealloc就是对象的遗言  
一旦重写了dealloc方法,就必须调用[super dealloc],并且放在最后面调用  
使用注意  
**不能直接调用dealloc方法**  
一旦对象被回收了,它占用的内存就不再可用,坚持使用会导致程序崩溃（野指针错误）

##### Xcode开启僵尸对象的监控



截图1.png

##### 苹果官方规定的内存管理原则

谁创建谁release:如果你通过alloc、new或[mutable]copy来创建一个对象，那么你必须调用release或autorelease。  
谁retain谁release:只要你调用了retain，就必须调用一次release。  
总结一下就是  
**有加就有减**  
**曾经让对象的计数器+1，就必须在最后让对象计数器-1**

##### set方法的内存管理

- (void)setCar:(Car\*)car { if(car!= \_car) { // 对当前正在使用的车（旧车）做一次release [\_car release]; // 对新车做一次retain操作 \_car=[car retain]; } }

##### dealloc方法内存管理

- (void)dealloc { //当人不在了，代表不用车了 // 对车做一次release操作 [\_car release]; //调用dealloc方法时候一定要调用父类的dealloc的方法 [super dealloc]; }

##### @property参数修饰

##### 控制set方法的内存管理

retain：release旧值，retain新值（用于OC对象）  
assign：直接赋值，不做任何内存管理(默认，用于非OC对象类型)  
copy ：release旧值，copy新值（一般用于NSString\*,block）

##### 控制需不需生成set方法

readwrite：同时生成set方法和get方法（默认）  
readonly ：只会生成get方法

##### 多线程管理

atomic ：性能低（默认）  
nonatomic：性能高

##### 控制set方法和get方法的名称

setter：设置set方法的名称，一定有个冒号:  
getter：设置get方法的名称

##### 强引用strong与弱引用weak的广义区别：

强引用**strong**也就是我们通常所讲的引用，其存亡直接决定了所指对象的存亡。如果不存在指向一个对象的引用，并且此对象不再显示列表中，则此对象会被从内存中释放。  
弱引用**weak**除了不决定对象的存亡外，其他与强引用相同。即使一个对象被持有无数个若引用，只要没有强引用指向他，那麽其还是会被清除。没办法，还是 “强哥” 有面子。

**简单讲strong等同retain**  
ARC时候用strong,MRC时候用return  
weak比assign多了一个功能，当对象消失后自动把指针变成nil，好处不言而喻。所以，我觉得在delegate时候用weak会好过assign。

##### 至于block的话，就要结合考虑。（weak，copy，assign）这三个结合情况，在下一篇的介绍。

**\_\_weak** ,**\_\_strong** ,**\_\_unsafe\_unretained**, **\_\_autoreleasing**用来修饰变量.  
(\_\_strong) 是缺省的关键词。  
(\_\_weak) 声明了一个可以自动 nil 化的弱引用。  
(\_\_unsafe\_unretained) 声明一个弱应用，但是不会自动nil化，也就是说，如果所指向的内存区域被释放了，这个指针就是一个野指针了。  
(\_\_autoreleasing) 用来修饰一个函数的参数，这个参数会在函数返回的时候被自动释放。

###### @class和#import的区别

作用上的区别  
**#import**会包含引用类的所有信息(内容), 包括引用类的变量和方法  
**@class**仅仅是告诉编译器有这么一个类,具体这个类里有什么信息, 完全不知道。

效率上的区别  
如果有上百个头文件都#import了同一个文件，或者这些文件依次被#import,那么一旦最开始的头文件稍有改动，后面引用到这个文件的所有类都需要重新编译一遍,编译效率非常低  
相对来讲，使用@class方式就不会出现这种问题了

##### MRC循环retian

循环retain的场景  
比如A对象retain了B对象，B对象retain了A对象

循环retain的弊端  
这样会导致A对象和B对象永远无法释放

循环retain的解决方案  
当两端互相引用时，应该一端用retain、一端用assign

**同理，在ARC下，如果两个对象相互引用，并且用strong修饰，那两个对象都无法释放。解决方案：其中一个对象用weak修饰，一个用strong修饰。那就可以让两个对象都可以释放。**

##### nil Nil NULL NSNull 之间的区别

**nil** 是对objective c id 对象赋空值  
**Nil**: 表示对类进行赋空值  
**NULL**: 用于对非对象指针赋空值，比如C指针  
**NSNull** 对于像NSArray这样的类型，nil或NULL不能做为加到其中的Object，如果定义了一个NSArray，为其分配了内存，又想设置其中的内容为空，则可以用[NSNULL null】返回的对象来初始化NSArray中的内容

# 数据结构

# 设计模式

## 设计原则

### 单一职责原则

一个类只负责一个功能领域中相应的职责

### 开闭原则

一个类或者软件实体尽量实现对扩展开发，对修改关闭

### 里氏代换

所有引用基类（父类）的地方，都能够使用子类的对象

### 依赖倒转

具体依赖抽象，而不是抽象依赖具体。

### 接口隔离

### 合成复用

### 迪米特法则

## 创建型

# 行为型

## 策略模式

### 组成成分：

1. 抽象策略类
2. 具体策略类
3. 策略上下文

# 牛人技术博客地址

1. <http://www.henishuo.com/ios-block-memory-cycle/?utm_source=tuicool&utm_medium=referral>
2. <http://www.cnblogs.com/mjios/p/4409853.html>
3. <https://blog.cnbluebox.com/blog/2014/07/01/cocoashen-ru-xue-xi-nsoperationqueuehe-nsoperationyuan-li-he-shi-yong/>
4. <http://www.jianshu.com/p/c47c24ab1e76>
5. <http://weibo.com/5612984599/profile?rightmod=1&wvr=6&mod=personinfo&is_hot=1#_loginLayer_1464445659010>
6. <http://casatwy.com/zen-yao-mian-shi-jia-gou-shi.html>
7. <http://blog.cnbluebox.com>
8. <http://my.oschina.net/u/566401/blog/219568>
9. <http://blog.sunnyxx.com/archives/page/3/>
10. <http://my.oschina.net/caijunrong/blog/512372>
11. <http://www.jianshu.com/users/256fb15baf75/latest_articles>
12. <http://www.jianshu.com/users/0a7ce85a2708/latest_articles>