IOS  面试题

Acorld

1.NSNotificationCenter 是同步的，不能操作耗时任务

2.KVO，是保存监听对象object，继承并重写被监听对象的set方法，将object的isa指针指向子类，同步，单线程。

3. CAAnimation及其子类的用途

CAAnimation : NSObject

CAPropertyAnimation : CAAnimation

CABasicAnimation : CAPropertyAnimation //平移、缩放、旋转，不改变位置

CAKeyframeAnimation : CAPropertyAnimation//关键帧动画-路径动画（values/path）

CATransition : CAAnimation//转场动画，type

CAAnimationGroup : CAAnimation//组合动画，容器

注：CAPropertyAnimation也是不能直接使用的，也要使用它的子类.

所以，能用的动画类只剩下4个：CABasicAnimation、CAKeyframeAnimation、CATransition、CAAnimationGroup

4. 谓词

== %@

MTATCH 正则

CONTAIN %@

BEGINSWITH %@

ENDSWITH %@

IN {‘AA’,’BB’}

BETWEEN {1,5}

LIKE \*%@\*

5.Core Graphics

主要是点，线，路径，图形图像的绘制

6进程与线程

两者都是程序运行基本单元。

不同点：

进程有单独的地址空间，多进程效率优于多线程，但进程切换耗费资源

线程是进程上的不同执行路径，一个线程死掉，整个进程死掉，线程可以共享变量和地址

7.block

block最需注意的是ARC与MARC的使用

NSGlobalBlock：类似函数，位于text段；与是否arc无关-未引用外部变量即为NSGlobalBlock，可以当做函数使用。

NSStackBlock：位于栈内存，函数返回后Block将无效；

NSMallocBlock：位于堆内存。

* Block\_copy与copy等效，Block\_release与release等效；
* 对Block不管是retain、copy、release都不会改变引用计数retainCount，retainCount始终是1；
* NSGlobalBlock：retain、copy、release操作都无效；
* NSStackBlock：retain、release操作无效，必须注意的是，NSStackBlock在函数返回后，Block内存将被回收。即使retain也没用。容易犯的错误是[[mutableAarry addObject:stackBlock]，（补：在arc中不用担心此问题，因为arc中会默认将实例化的block拷贝到堆上）在函数出栈后，从mutableAarry中取到的stackBlock已经被回收，变成了野指针。正确的做法是先将stackBlock copy到堆上，然后加入数组：[mutableAarry addObject:[[stackBlock copy] autorelease]]。支持copy，copy之后生成新的NSMallocBlock类型对象。
* NSMallocBlock支持retain、release，虽然retainCount始终是1，但内存管理器中仍然会增加、减少计数。copy之后不会生成新的对象，只是增加了一次引用，类似retain；

尽量不要对Block使用retain操作。

8.浅拷贝与深拷贝

前者拷贝指针，只增加引用计数

后者拷贝内容，新建相同引用计数的对象

copy中除不可变对象的copy方法外，都为深拷贝（新建对象）（mutablecopy）

9. 写一个foo-bar逻辑判断小程序。（从1-100遍历一遍，如果是3的倍数，输出foo，如果是5的倍数，输出bar，如果是3和5的倍数，输出foobar。

- (void)fooBar

{

for (int i = 1; i < 101; i ++)

{

if (i % 15 ==0) NSLog(@"foobar:%d",i);

else if (i % 3 == 0) NSLog(@"foo:%d",i);

else if(i % 5 == 0) NSLog(@"bar:%d",i);

else continue;

}

}

10.void\* 和 id

marc

可以互相转换，无压力

arc时，需要\_\_bridge（\_\_unsafe\_unretained）来转换

\_\_bridge\_retained 增加引用

\_\_bridge\_transfer 转换所有权

11 UIView(UIResponder)和CALayer(NSObject)

前者管理并显示后者，主要负责事件响应

后者负责绘制渲染后者

12.  hitTest、pointInside及如何修改事件响应的顺序。

Touch🡪 UIApplication 事件队列🡪UIApplication取出事件传递给key window🡪window调用hitTest:withEvent:寻找touch的view。

寻找touch的view顺序：

window顶级view调用hitTest:wiithEvent:，该方法首先调用pointInside:withEvent:,如果返回NO，则hitTest:wiithEvent:返回nil，遍历window下一层级的view;否则，遍历该view的所有子视图发送hitTest:wiithEvent:，如果pointInside:withEvent:返回YES，则停止遍历，返回该子视图，否则继续操作，如果所有子视图pointInside:withEvent: 都返回NO,则hitTest:wiithEvent返回view本身。

13.NSOperation

使用方法：

一、两种定义好的子类：

NSBlockOperation 通过block来执行方法

NSInvocationOperation通过selector来执行方法

二、新建子类：

是否支持并发：

是：start 注意不要调用[super start]；

isFinished

isExecuting

isConcurrent(ios5以后忽略，而用queue的maxConcurrentOperationCount判断)

否：main

14 Quatrz 2D 作为Core Graphics的核心部分

上下文，路径，状态

15 Runloop 事件分发机制的一个实现。就是一个事件处理的循环,用来不停的调度工作以及处理输入事件。

合适的时候醒来做合适的事（做周期性或者定时器）

end\_ = NO;

[NSThread detachNewThreadSelector:@selector(runOnNewThread) toTarget:self withObject:nil];

while (!end\_)

{

[[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode beforeDate:[NSDate distantFuture]];

}

- (void)runOnNewThread

{

//需要在主线程中更新end\_的值-唤醒runloop，否则就等其他操作唤醒runloop

[self performSelectorOnMainThread:@selector(setEnd) withObject:nil waitUntilDone:NO];

}

- (void)setEnd

{

end\_ = YES;

}

16 NSInvocation

消息分发

- (void)testInvocation

{

NSMethodSignature \*sig = [[self class] instanceMethodSignatureForSelector:@selector(hello:y:)];

NSInvocation \*invocation = [NSInvocation invocationWithMethodSignature:sig];

[invocation setTarget:self];

[invocation setSelector:@selector(hello:y:)];

NSInteger x = 2;

NSInteger y = 3;

[invocation setArgument:&x atIndex:2];

[invocation setArgument:&y atIndex:3];

[invocation retainArguments];

[invocation invoke];

NSString \*result;

[invocation getReturnValue:&result];

}

17 ios8

18 在哪块是专家？

19.GCD

**dispatch\_group\_async**

将线程放到group中执行，执行完毕，group\_notify通知

**dispatch\_barrier\_async**

栅栏，先执行前面的线程，后执行后面的线程

20.dispatch\_queue\_create

dispatch\_queue\_create 用于创建用户线程队列。可以创建Serial/Concurrent Dispatch Queue 两种队列，即串行与并行队列.

* 决不能随意的大量生产Serial Dispatch Queue。
* Concurrent Dispatch Queue不过创建多少都没有问题，因为Concurrent Dispatch Queue所使用的线程由系统的XNU内核高效管理，不会影响系统性能。
* OSX 8以前，需要**dispatch\_release()手动释放。**
* **对于串行队列，每创建一个串行队列，系统就会对应创建一个线程，同时这些线程都是并行执行的，只是在串行队列中的任务是串行执行的。大量的创建串行队列会导致大量消耗内存，这是不可取的做法。串行队列的优势在于他是一个线程，所以在操作一个全局数据时候是线程安全的。当想并行执行而不发生数据竞争时候可以用并行队列操作**

21.sdk 常用设计模式 ：

\* 监听者模式

\* 类簇模式

\* 工厂模式

\* MVC模式

\* 单例模式