多线程

你们项目中为什么多线程用GCD而不用NSOperation呢？你有没有发现国外的大牛他们多线程都是用NSOperation？你能告诉我他们这样做的理由吗？

关系:

①:先搞清两者的关系,NSOpertaionQueue用GCD构建封装的，是GCD的高级抽象!

②:GCD仅仅支持FIFO队列，而NSOperationQueue中的队列可以被重新设置优先级，从而实现不同操作的执行顺序调整。GCD不支持异步操作之间的依赖关系设置。如果某个操作的依赖另一个操作的数据（生产者-消费者模型是其中之一），使用NSOperationQueue能够按照正确的顺序执行操作。GCD则没有内建的依赖关系支持。

③:NSOperationQueue支持KVO，意味着我们可以观察任务的执行状态。

了解以上不同，我们可以从以下角度来回答

性能:①:GCD更接近底层，而NSOperationQueue则更高级抽象，所以GCD在追求性能的底层操作来说，是速度最快的。这取决于使用Instruments进行代码性能分析，如有必要的话

②:从异步操作之间的事务性，顺序行，依赖关系。GCD需要自己写更多的代码来实现，而NSOperationQueue已经内建了这些支持

③:如果异步操作的过程需要更多的被交互和UI呈现出来，NSOperationQueue会是一个更好的选择。底层代码中，任务之间不太互相依赖，而需要更高的并发能力，GCD则更有优势

最后的一句话:别忘了高德纳的教诲：“在大概97%的时间里，我们应该忘记微小的性能提升。过早优化是万恶之源。”只有Instruments显示有真正的性能提升时才有必要用低级的GCD。

详解GCD死锁

unix上进程怎么通信?

- UNIX主要支持三种通信方式：
- 基本通信：主要用来协调进程间的同步和互斥
    - 锁文件通信:通信的双方通过查找特定目录下特定类型的文件(称锁文件)来完成进程间 对临界资源访问时的互斥；例如进程p1访问一个临界资源，首先查看是否有一个特定类型文件，若有，则等待一段时间再查找锁文件。
    - 记录锁文件
- 管道通信：适应大批量的数据传递

- IPC ： 适应大批量的数据传递

列举几种进程的同步机制、进程的通信途径、死锁及死锁的处理方法。

- 进程的同步机制原子操作 信号量机制 自旋锁 管程，会合，分布式系统
- 进程之间通信的途径：共享存储系统消息传递系统管道：以文件系统为基础
- 进程死锁的原因：资源竞争及进程推进顺序非法
- 死锁的4个必要条件：互斥、请求保持、不可剥夺、环路
- 死锁的处理：鸵鸟策略、预防策略、避免策略、检测与解除死锁

线程与进程的区别和联系?

- 线程是进程的基本单位。
- 进程和线程都是由操作系统所产生的程序运行的基本单元,系统利用该基本单元实现系统对应用的并发性。
- 进程和线程的主要差别在于它们是不同的操作系统资源管理方式。
- 进程有独立的地址空间,一个进程崩溃后,在保护模式下 不会对其它进程产生影响。
- 线程只是一个进程中的不同执行路径。
- 线程有自己的堆栈和局部变量,但线程之间没有单独的地址空间,一个线程死掉就等于整个进程死掉,所以多进程的程序要比多线程的程序健壮,但在进程切换时,耗费资源较大,效率要差一些。
- 但对于一些要求同时进行并且又要共享某些变量的并发操作,只能用线程,不能用进程。

iOS线程间怎么通信?

- performSelector:onThread:withObject:waitUntilDone:
- NSMachPort
  (基本机制：A线程（父线程）创建NSMachPort对象，并加入A线程的runloop。当创建B线程（辅

助线程）时，将创建的NSMachPort对象传递到主体入口点，B线程（辅助线程）就可以使用相同的端口对象将消息传回A线程（父线程）。）

iOS多线程的底层实现?

- 首先搞清楚什么是线程、什么是多线程
  - Mach是第一个以多线程方式处理任务的系统，因此多线程的底层实现机制是基于Mach的线程
  - 开发中很少用Mach级的线程，因为Mach级的线程没有提供多线程的基本特征，线程之间是独立的
- 开发中实现多线程的方案
  - C语言的POSIX接口：#include <pthread.h>
  - OC的NSThread
  - C语言的GCD接口（性能最好，代码更精简）
  - OC的NSOperation和NSOperationQueue（基于GCD）

谈谈多线程安全问题的几种解决方案?何为线程同步,如何实现的?分线程回调主线程方法是什么,有什么作用?

- 解决方案: 使用锁: 锁是线程编程同步工具的基础。锁可以让你很容易保护代码中一大块区域以便你可以确保代码的正确性。
  1. 使用POSIX互斥锁;
  2. 使用NSLock类;
  3. 使用@synchronized指令等。
- 回到主线程的方法: dispatch_async(dispatch_get_main_queue(), ^{ });
- 作用: 主线程是显示UI界面,子线程多数是进行数据处理

使用atomic一定是线程安全的吗?

```
不是的。
atomic原子操作，系统会为setter方法加锁。 具体使用 @synchronized(self){//code }
nonatomic不会为setter方法加锁。
atomic：线程安全，需要消耗大量系统资源来为属性加锁
nonatomic：非线程安全，适合内存较小的移动设备
使用atomic并不能保证绝对的线程安全，对于要绝对保证线程安全的操作，还需要使用更高级的方式来处理，比如NSSpinLock、@
```

谈谈你对多线程开发的理解(多线程的好处，多线程的作用)？ios中有几种实现多线程的方法?

- 好处：
  - 使用线程可以把占据时间长的程序中的任务放到后台去处理
  - 用户界面可以更加吸引人，这样比如用户点击了一个按钮去触发某些事件的处理，可以弹出一个进度条来显示处理的进度
  - 程序的运行效率可能提高
  - 在一些等待的任务实现上如用户输入、文件读写和网络收发数据等，线程就比较有用了。
- 缺点：
  - 如果有大量的线程,会影响性能,因为操作系统需要在它们之间切换。
  - 更多的线程需要更多的内存空间。
  - 线程的中止需要考虑其对程序运行的影响。
- 通常块模型数据是在多个线程间共享的，需要防止线程死锁情况的发生。
- 实现多线程的方法：
  1. NSObject的类方法 // -(void)performSelectornBackground/OnMainThread:(SEL)aSelector withObject:(id)arg
  2. NSThread
  3. NSOperation
  4. GCD

OC中异步使用的哪种事件模型,iOS中异步实现机制

- 异步非阻塞 I/O（AIO）

详细谈谈GCD

1. 推出的时间 iOS4 目的是用来取代NSThread（ios2.0推出）的，是 C语言框架，它能够自动利用更多CPU的核数，并且会自动管理线程的生命周期。
   - CGD的两个核心概念：任务， 队列
   - 任务：记为在block中执行的代码。
   - 队列：用来存放任务的。

- 注意事项： 队列 != 线程。队列中存放的任务最后都要由线程来执行!。队列的原则:先进先出,后进后出(FIFO/ First In First Out)

2. 队列又分为四种种：1 串行队列 2 并发队列 3 主队列 4 全局队列

- 串行队列： 任务一个接一个的执行。
- 并发队列： 队列中的任务并发执行。
- 主队列： 跟主线程相关的队列，主队列里面的内容都会在主线程中执行（我们一般在主线程中刷新UI）。
- 全局队列： 一个特殊的并发队列。

3. 并发队列与全局队列的区别：

- 并发队列有名称,可以跟踪错误。全局队列没有
- 在ARC中两个队列不需要考虑释放内存,但是在MRC中并发队列是创建出来的需要release操作，而全局队列只有一个不需要。
- 一般在开发过程中我们使用全局队列。

4. 执行任务的两个函数

- '同步'执行任务:dispatch_sync(<#dispatch_queue_t queue#>, <#^(void)block#>)
- '异步'执行任务:dispatch_async(dispatch_queue_t queue, <#^(void)block#>)

5. "同步"和"异步"的区别:

- "同步": 只能在'当前'线程中执行任务,不具备开启新线程的能力.
- "异步": 可以在'新'的线程中执行任务,具备开启新线程的能力.

6. 各个队列的执行效果：

- 串行队列同步执行，既在当前线程中顺序执行
- 串行队列异步执行，开辟一条新的线程，在该线程中顺序执行
- 并行队列同步执行，不开辟线程，在当前线程中顺序执行
- 并行队列异步执行，开辟多个新的线程，并且线程会重用，无序执行
- 主队列异步执行，不开辟新的线程，顺序执行
- 主队列同步执行，会造成死锁（'主线程'和'主队列'相互等待,卡住主线程）

7. 线程间通讯：经典案例：子线程进行耗时操作（例如下载更新等）主线程进行UI刷新。

   - 经典用法(子线程下载(耗时操作),主线程刷新UI):

```
dispatch_async(dispatch_get_global_queue(0, 0), ^{

 // 执行耗时的异步操作...

dispatch_async(dispatch_get_main_queue(), ^{

// 回到主线程，执行UI刷新操作
```

8. 延迟操作

   调用 NSObject 方法:[self performSelector:@selector(run) withObject:nil afterDelay:2.0];

   // 2秒后再调用self的run方法

   GCD函数实现延时执行:dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(2.0 * NSEC_PER_SEC)), dispatch_get_main_queue(), ^{

   // 2秒后执行这里的代码... 在哪个线程执行，跟队列类型有关

9. 队列组的使用：

   - 项目需求:首先:分别异步执行两个耗时操作;其次:等两次耗时操作都执行完毕后,再回到主线程执行操作.使用队列组(dispatch_group_t)快速,高效的实现上述需求.

   dispatch_group_t group = dispatch_group_create(); // 队列组

   dispatch_queue_t queue = dispatch_get_global_queue(0, 0); // 全局并发队列

   dispatch_group_async(group, queue, ^{// 异步执行操作1

   // longTime1

   });

   dispatch_group_async(group, queue, ^{ // 异步执行操作2

```
// longTime2

});

dispatch_group_notify(group, dispatch_get_main_queue(), ^{

// 在主线程刷新数据
// reload Data
});
```

GCD内部怎么实现的

- iOS和OS X的核心是XNU内核，GCD是基于XNU内核实现的
- GCD的API全部在libdispatch库中
- GCD的底层实现主要有Dispatch Queue和Dispatch Source
  - Dispatch Queue： 管理block(操作)
  - Dispatch Source： 处理事件(MACH端口发送,MACH端口接收,检测与进程相关事件等10种事件)

GCD的queue、main queue中执行的代码一定是在main thread么?

- 对于queue中所执行的代码不一定在main thread中。如果queue是在主线程中创建的，那么所执行的代码就是在主线程中执
- 对于main queue就是在主线程中的，因此一定会在主线程中执行。获取main queue就可以了，不需要我们创建，获取方式通

如何用GCD同步若干个异步调用？（如根据若干个url异步加载多张图片，然后在都下载完成后合成一张整图）

```
使用Dispatch Group追加block到Global Group Queue,这些block如果全部执行完毕，就会执行Main Dispatch Queue中的结

dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, );
dispatch_group_t group = dispatch_group_create();
dispatch_group_async(group, queue, ^{ /*加载图片1 */ });
dispatch_group_async(group, queue, ^{ /*加载图片2 */ });
dispatch_group_async(group, queue, ^{ /*加载图片3 */ });
dispatch_group_notify(group, dispatch_get_main_queue(), ^{
        // 合并图片
});
```

有a、b、c、d 4个异步请求，如何判断a、b、c、d都完成执行？如果需要a、b、c、d顺序执行，该如何实现？

```
1.  对于这四个异步请求，要判断都执行完成最简单的方式就是通过GCD的group来实现：
2.  dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
3.  dispatch_group_t group = dispatch_group_create();
4.  dispatch_group_async(group, queue, ^{ /*任务a */ });
5.  dispatch_group_async(group, queue, ^{ /*任务b */ });
6.  dispatch_group_async(group, queue, ^{ /*任务c */ });
7.  dispatch_group_async(group, queue, ^{ /*任务d */ });
8.
9.  dispatch_group_notify(group, dispatch_get_main_queue(), ^{
10.     // 在a、b、c、d异步执行完成后，会回调这里
11. });
当然，我们还可以使用非常老套的方法来处理，通过四个变量来标识a、b、c、d四个任务是否完成，然后在runloop中让其等待，要求顺序执行，那么可以将任务放到串行队列中，自然就是按顺序来异步执行了
```

发送10个网络请求，然后再接收到所有回应之后执行后续操作，如何实现？

```
从题目分析可知，10个请求要全部完成后，才执行某一功能。比如，下载10图片后合成一张大图，就需要异步全部下载完成后，才
做法：通过dispatch_group_t来实现，将每个请求放入到Group中，将合并成大图的操作放在dispatch_group_notify中实现。
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch_group_t group = dispatch_group_create();
dispatch_group_async(group, queue, ^{ /*加载图片1 */ });
dispatch_group_async(group, queue, ^{ /*加载图片2 */ });
dispatch_group_async(group, queue, ^{ /*加载图片3 */ });
dispatch_group_notify(group, dispatch_get_main_queue(), ^{
    // 合并图片
});
```

苹果为什么要废弃dispatch_get_current_queue?

- dispatch_get_current_queue容易造成死锁。详情点击该API查看官方注释。

如果让你来实现 dispatch_once，你会怎么做?

- http://www.dreamingwish.com/article/gcd-guide-dispatch-once-2.html（超级详细解析）
- 个人觉得说出实现的思路即可，无锁的线程同步编程，每一处的线程竞争都考虑到并妥善处理
- 线程A执行Block时，任何其它线程都需要等待。
- 线程A执行完Block应该立即标记任务完成状态，然后遍历信号量链来唤醒所有等待线程。

- 线程A遍历信号量链来signal时，任何其他新进入函数的线程都应该直接返回而无需等待。

- 线程A遍历信号量链来signal时，若有其它等待线程B仍在更新或试图更新信号量链，应该保证此线程B能正确完成其任务：a.直接返回 b.等待在信号量上并很快又被唤醒。

- 线程B构造信号量时，应该考虑线程A随时可能改变状态（“等待”、“完成”、“遍历信号量链”）。

- 线程B构造信号量时，应该考虑到另一个线程C也可能正在更新或试图更新信号量链，应该保证B、C都能正常完成其任务：a.增加链节并等待在信号量上 b.发现线程A已经标记“完成”然后直接销毁信号量并退出函数。

关于NSOperation:

- NSOperation: 抽象类,不能直接使用,需要使用其子类.(类似的类还有核心动画)
- 两个常用子类: NSInvocationOperation(调用) 和 NSBlockOperation(块);
- 两者没有本质区别,后者使用 Block 的形式组织代码,使用相对方便.
- NSInvocationOperation在调用start方法后，不会开启新的线程只会在当前线程中执行。
- NSBlockOperation 在调用start方法后，如果封装的操作数>1会开辟多条线程执行 =1 只会在当前线程执行.
- NSOperationQueue 创建的操作队列默认为全局队列，队列中的操作执行顺序是无序的，如果需要让它有序执行需要添加依赖关系。
- // 操作op3依赖于操作op2 [op3 addDependency:op2];
- // 操作op2依赖于操作op1 [op2 addDependency:op1];
- 同时可以设置最大并发数
- NSOperationQueue NSOperation支持 取消暂停的操作 但是正在进行的的操作并不能取消，这一旦取消不可恢复.
- NSOperationQueue支持KVO，可以监测operation是否正在执行（isExecuted）、是否结束（isFinished），是否取消（isCanceld）

NSOperation queue?

- 存放NSOperation的集合类。不能说队列，不是严格的先进先出

NSOperation与GCD的区别

- GCD

1. GCD是iOS4.0推出的，主要针对多核cpu做了优化，是纯c语言的技术。
2. GCD是将任务（block）添加到队列（串行、并行、全局、主队列），并且以同步/异步的方式执行任务的函数。
3. GCD提供了一些NSOperation不具备的功能

    - 一次性执行
    - 延迟执行
    - 调度组
    - GCD 是严格的队列，先进先出 FIFO；

- NSOperation

1. NSOperation是iOS2.0推出的，iOS4.0以后又重写了NSOperation
2. NSOperation是将操作（异步的任务）添加到队列（并发队列），就会执行指定的函数
3. NSOperation提供的方便操作

    - 最大并发数
    - 队列的暂停和继续
    - 取消所有的操作
    - 指定操作之间的依赖关系依赖关系，可以让异步任务同步执行.
    - 将KVO用于NSOperation中，监听一个operation是否完成。
    - 能够设置NSOperation的优先级，能够使同一个并行队列中的任务区分先后地执行
    - 对NSOperation进行继承，在这之上添加成员变量与成员方法，提高整个代码的复用度

GCD与NSThread的区别

- NSThread 通过 @selector 指定要执行的方法，代码分散
- GCD 通过 block 指定要执行的代码，代码集中, 所有的代码写在一起的，让代码更加简单，易于阅读和维护
- 使用 GCD 不需要管理线程的创建/销毁/复用的过程！程序员不用关心线程的生命周期
- 如果要开多个线程 NSThread 必须实例化多个线程对象
- NSThread 靠 NSObject 的分类方法实现的线程间通讯，

为什么要取消/恢复队列呢?

- 一般在内存警告后取消队列中的操作。
- 为了保证scorllView在滚动的时候流畅 通常在滚动开始时，暂停队列中的所有操作，滚动结束后，恢复操作。

Object C中创建线程的方法是什么?如果在主线程中执行代码，方法是什么?如果想延时执行代码、方法又是什么?

```
线程创建有三种方法：使用NSThread创建、使用GCD的dispatch、使用子类化的NSOperation,然后将其加入NSOperationQueue

NSThread创建线程的三种方法：
 NSThread *thread = [[NSThread alloc]initWithTarget:self selector:@selector(run:) object:@"nil"];
 [NSThread detachNewThreadSelector:@selector(run:) toTarget:self withObject:@"我是分离出来的子线程"];
[self performSelectorInBackground:@selector(run:) withObject:@"我是后台线程"];

在主线程执行代码，就调用performSelectorOnMainThread方法。

如果想延时执行代码可以调用performSelector:onThread:withObject:waitUntilDone:方法；

GCD：
利用异步函数dispatch_async()创建子线程。

在主线程执行代码，dispatch_async(dispatch_get_main_queue(), ^{});

延迟执行代码（延迟.可以控制代码在哪个线程执行）：
dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(2.0 * NSEC_PER_SEC)), dispatch_get_global_queu

NSOperationQueue：
使用NSOperation的子类封装操作，再将操作添加到NSOperationQueue创建的队列中，实现多线程。
在主线程执行代码，只要将封装代码的NSOperation对象添加到主队列就可以了。
```

下面关于线程管理错误的是

```
A. GCD所用的开销要比NSThread大
B. 可以在子线程中修改UI元素
C. NSOperationQueue是比NSthread更高层的封装
D. GCD可以根据不同优先级分配线程
```

- 参考答案：B
- 理由：首先，UI元素的更新必须在主线程。GCD与Block配合使用，block需要自动捕获上下文变量信息等，因此需要更多的资源，故比NSThread开销要大一些。NSOperationQueue与NSOperation配合使用，比NSThread更易于操作线程。GCD提供了多个优先级，我们可以根据设置优先级，让其自动为我们分配线程。

文章如有问题，请留言，我将及时更正。