# 1. 『GCD』详尽总结

Grand Central Dispatch(GCD) 。它主要用于优化应用程序以支持多核处理器以及其他对称多处理系统。它是一个在线程池模式的基础上执行的并发任务。基于C语言的线程管理方案

1) GCD 好处

* GCD 可用于多核的并行运算
* GCD 会自动利用更多的 CPU 内核（比如双核、四核）
* GCD 会自动管理线程的生命周期（创建线程、调度任务、销毁线程）
* 程序员只需要告诉 GCD 想要执行什么任务，不需要编写任何线程管理代码

2) GCD 任务: 就是执行操作的意思

* 同步执行（sync）同步添加任务到指定的队列中，在添加的任务执行结束之前，会一直等待，直到队列里面的任务完成之后再继续执行。

只能在当前线程中执行任务，不具备开启新线程的能力。

* 异步执行（async）异步添加任务到指定的队列中，它不会做任何等待，可以继续执行任务。

可以在新的线程中执行任务，具备开启新线程的能力。

注意：异步执行（async）虽然具有开启新线程的能力，但是并不一定开启新线程。这跟任务所指定的队列类型有关

3) GCD 队列：指执行任务的等待队列，即用来存放任务的队列。队列是一种特殊的线性表，采用 FIFO（先进先出）的原则

 串行队列（Serial Dispatch Queue）：

每次只有一个任务被执行。让任务一个接着一个地执行。（只开启一个线程，一个任务执行完毕后，再执行下一个任务）

 并发队列（Concurrent Dispatch Queue）

可以让多个任务并发（同时）执行。（可以开启多个线程，并且同时执行任务）  
注意：并发队列的并发功能只有在异步（dispatch\_async）函数下才有效

3.1 ）队列的创建方法/获取方法：

使用dispatch\_queue\_create来创建队列，需要传入两个参数，第一个参数表示队列的唯一标识符，用于 DEBUG，可为空，Dispatch Queue 的名称推荐使用应用程序 ID 这种逆序全程域名；第二个参数用来识别是串行队列还是并发队列。

DISPATCH\_QUEUE\_SERIAL 表示串行队列，可以直接写成NULL DISPATCH\_QUEUE\_CONCURRENT 表示并发队列。

  
3.2 ）主队列（Main Dispatch Queue）： 特殊的串行队列

·所有放在主队列中的任务，都会放到主线程中执行。

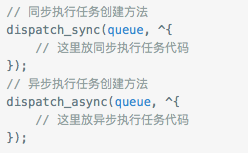
Macintosh HD:Users:admin:Desktop:屏幕快照 2018-04-08 14.55.41.png

3.3）全局并发队列（Global Dispatch Queue）:全局并发队列可以作为普通并发队列来使用

可以使用dispatch\_get\_global\_queue来获取。需要传入两个参数。第一个参数表示队列优先级，一般用DISPATCH\_QUEUE\_PRIORITY\_DEFAULT。第二个参数暂时没用，用0即可

Macintosh HD:Users:admin:Desktop:屏幕快照 2018-04-08 14.55.50.png

3.4）任务的创建方法



4）GCD的基本使用



4.1）同步执行 + 并发队列：

·所有任务都是在当前线程（主线程）中执行的，没有开启新的线程（同步执行不具备开启新线程的能力）

·任务按顺序执行的。按顺序执行的原因：虽然并发队列可以开启多个线程，并且同时执行多个任务。但是因为本身不能创建新线程，只有当前线程这一个线程（同步任务不具备开启新线程的能力），所以也就不存在并发。而且当前线程只有等待当前队列中正在执行的任务执行完毕之后，才能继续接着执行下面的操作（同步任务需要等待队列的任务执行结束）。所以任务只能一个接一个按顺序执行，不能同时被执行。

特点：在当前线程中执行任务，不会开启新线程，执行完一个任务，再执行下一个任务

4.2）异步执行 + 并发队列

特点：可以开启多个线程，任务交替（同时）执行。

4.3）同步执行 + 串行队列

特点：不会开启新线程，在当前线程执行任务。任务是串行的，执行完一个任务，再执行下一个任务。

4.4）异步执行 + 串行队列

特点：会开启新线程，但是因为任务是串行的，执行完一个任务，再执行下一个任务。

4.5）同步执行 + 主队列

* 特点(主线程调用)：互等卡主不执行。(主队列中的任务必须按顺序挨个执行, 任务1要等主线程有空的时候（即主队列中的所有任务执行完）才能执行, 主线程要执行完“打印end”的任务后才有空, “任务1”和“打印end”两个任务互相等待，造成死锁)

特点(其他线程调用)：不会开启新线程，执行完一个任务，再执行下一个任务。

4.6）异步执行 + 主队列

·所有任务都是在当前线程（主线程）中执行的，并没有开启新的线程（虽然异步执行具备开启线程的能力，但因为是主队列，所以所有任务都在主线程中）

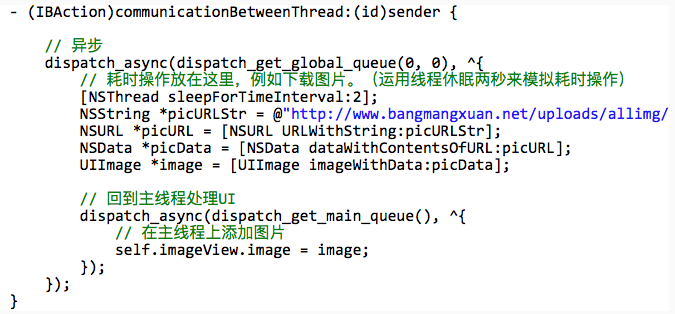
·所有任务是在打印的syncConcurrent---begin和syncConcurrent---end之后才开始执行的（异步执行不会做任何等待，可以继续执行任务）

·任务是按顺序执行的（因为主队列是串行队列，每次只有一个任务被执行，任务一个接一个按顺序执行）

特点：只在主线程中执行任务，执行完一个任务，再执行下一个任务

5 ）GCD 线程间的通信

我们通常把一些耗时的操作放在其他线程，比如说图片下载、文件上传等耗时操作。而当我们有时候在其他线程完成了耗时操作时，需要回到主线程，那么就用到了线程之间的通讯。



6）GCD 栅栏方法：dispatch\_barrier\_async

特点：在执行完栅栏前面的操作之后，才执行栅栏操作，最后再执行栅栏后边的操作

7）GCD延时执行方法：dispatch\_after



8）GCD 一次性代码（只执行一次）：dispatch\_once

我们在创建单例、或者有整个程序运行过程中只执行一次的代码时，我们就用到了 GCD 的 dispatch\_once 函数。使用

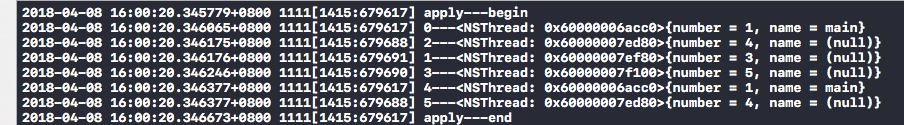
dispatch\_once 函数能保证某段代码在程序运行过程中只被执行1次，并且即使在多线程的环境下，dispatch\_once也可以保证线程安全。



9）GCD 快速迭代方法：dispatch\_apply

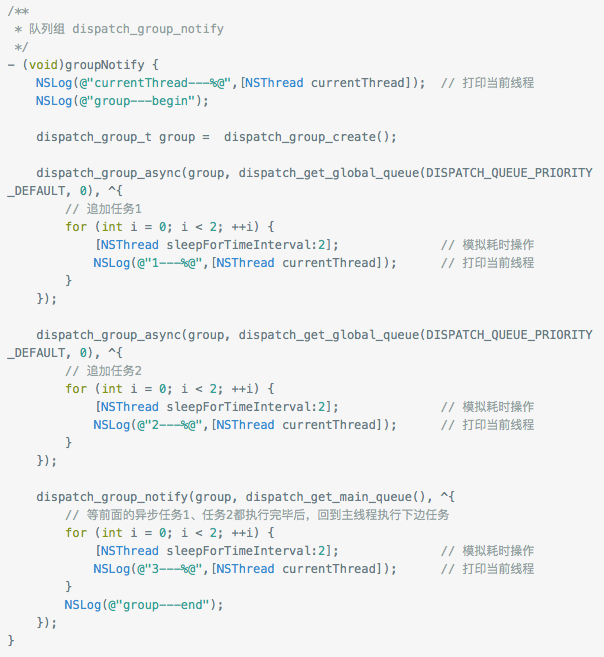
如果是在串行队列中使用 dispatch\_apply，那么就和 for 循环一样，按顺序同步执行，可以利用并发队列进行异步执行。比如说遍历 0~5 这6个数字，for 循环的做法是每次取出一个元素，逐个遍历。dispatch\_apply 可以 在多个线程中同时（异步）遍历多个数字。无论是在串行队列，还是异步队列中，dispatch\_apply 都会等待全部任务执行完毕，这点就像是同步操作，也像是队列组中的 dispatch\_group\_wait方法。





10) GCD 队列组：dispatch\_group

有时候我们会有这样的需求：分别异步执行2个耗时任务，然后当2个耗时任务都执行完毕后再回到主线程执行任务。这时候我们可以用到 GCD 的队列组。



·dispatch\_group\_notify: 监听 group 中任务的完成状态，当所有的任务都执行完成后，追加任务到 group 中，并执行任务。

·暂停当前线程（阻塞当前线程），等待指定的 group 中的任务执行完成后，才会往下继续执行 dispatch\_group\_wait(group, DISPATCH\_TIME\_FOREVER);

11）GCD 信号量：dispatch\_semaphore

就是一种可用来控制访问资源的数量的标识，设定了一个信号量，在线程访问之前，加上信号量的处理，则可告知系统按照我们指定的信号量数量来执行多个线程。

实际开发中主要用于：

·保持线程同步，将异步执行任务转换为同步执行任务

·保证线程安全，为线程加锁



输出结果:

currentThread---<NSThread: 0x60400006bc80>{number = 1, name = main}

semaphore---begin

1---<NSThread: 0x600000272300>{number = 3, name = (null)}

semaphore---end,number = 100

semaphore---end 是在执行完 number = 100; 之后才打印的。而且输出结果 number 为 100。这是因为异步执行不会做任何等待，可以继续执行任务。异步执行将任务1追加到队列之后，不做等待，接着执行dispatch\_semaphore\_wait方法。此时 semaphore == 0，当前线程进入等待状态。然后，异步任务1开始执行。任务1执行到dispatch\_semaphore\_signal之后，总信号量，此时 semaphore == 1，dispatch\_semaphore\_wait方法使总信号量减1，正在被阻塞的线程（主线程）恢复继续执行。最后打印semaphore---end,number = 100。这样就实现了线程同步，将异步执行任务转换为同步执行任务。

线程安全：如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。

若每个线程中对全局变量、静态变量只有读操作，而无写操作，一般来说，这个全局变量是线程安全的；若有多个线程同时执行写操作（更改变量），一般都需要考虑线程同步，否则的话就可能影响线程安全。

线程同步：可理解为线程 A 和 线程 B 一块配合，A 执行到一定程度时要依靠线程 B 的某个结果，于是停下来，示意 B 运行；B 依言执行，再将结果给 A；A 再继续操作。

场景：总共有50张火车票，有两个售卖火车票的窗口，一个是北京火车票售卖窗口，另一个是上海火车票售卖窗口。两个窗口同时售卖火车票，卖完为止。





可以看到在不考虑线程安全，不使用 semaphore 的情况下，得到票数是错乱的，这样显然不符合我们的需求，所以我们需要考线程安全问题。





可以看出，在考虑了线程安全的情况下，使用 dispatch\_semaphore机制之后，得到的票数是正确的，没有出现混乱的情况。我们也就解决了多个线程同步的问题。

Dispatch Semaphore 提供了三个函数:

dispatch\_semaphore\_create（信号量值）：创建一个Semaphore并初始化信号的总量，如果小于0则会返回NULL，信号量值表示，最多几个资源可访问

dispatch\_semaphore\_signal：发送一个信号，让信号总量加1

dispatch\_semaphore\_wait（信号量，等待时间）：可以使总信号量减1，当信号总量为0时就会一直等待（阻塞所在线程），否则就可以正常执行。

注意，正常的使用顺序是先降低然后再提高，这两个函数通常成对使用

注意：信号量的使用前提是：想清楚你需要处理哪个线程等待（阻塞），又要哪个线程继续执行，然后使用信号量。

GCD是一个任务调度系统，负责将我们提交的任务分配到不同的线程上去执行，我们提交任务的方式有同步和异步。

进程：可以理解成一个运行中的应用程序，是系统进行资源分配和调度的基本单位，是操作系统结构的基础，主要管理资源。

线程：是进程的基本执行单元，一个进程对应多个线程。

线程就像火车的一节车厢，进程则是火车。车厢（线程）离开火车（进程）是无法跑动的，而火车（进程）至少有一节车厢（主线程）。多线程可以看做多个车厢，它的出现是为了提高效率

第一种：pthread

特点：一套通用的多线程API，使用c，生命周期由程序员管理。几乎不用

第二种：NSThread

特点：1）使用更加面向对象 2）简单易用，可直接操作线程对象，使用oc，生命周期由程序员管理。几乎不用