Grand Central Dispatch (GCD) 是Apple开发的一个多核编程的较新的解决方法。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行任务。在Mac OS X 10.6雪豹中首次推出，也可在IOS 4及以上版本使用。

**1.为什么要用GCD呢？**

因为GCD有很多好处啊，具体如下：

* 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
* 既然GCD有这么多的好处，那么下面我们就来系统的学习一下GCD的使用方法。

**2. 任务和队列**

学习GCD之前，先来了解GCD中两个核心概念：任务和队列。

**任务**：就是执行操作的意思，换句话说就是你在线程中执行的那段代码。在GCD中是放在block中的。执行任务有两种方式：**同步执行**和**异步执行**。两者的主要区别是：是否具备开启新线程的能力。

* 同步执行（sync）：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）：可以在新的线程中执行任务，具备开启新线程的能力

**队列**：这里的队列指任务队列，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，即新任务总是被插入到队列的末尾，而读取任务的时候总是从队列的头部开始读取。每读取一个任务，则从队列中释放一个任务。在GCD中有两种队列：**串行队列**和**并发队列**。

* 并发队列（Concurrent Dispatch Queue）：可以让多个任务并发（同时）执行（自动开启多个线程同时执行任务）
  + 并发功能只有在异步（dispatch\_async）函数下才有效
* 串行队列（Serial Dispatch Queue）：让任务一个接着一个地执行（一个任务执行完毕后，再执行下一个任务）

## 3. GCD的使用步骤

GCD的使用步骤其实很简单，只有两步。

1. 创建一个队列（串行队列或并发队列）
2. 将任务添加到队列中，然后系统就会根据任务类型执行任务（同步执行或异步执行）

下边来看看队列的创建方法和任务的创建方法。

### 1. 队列的创建方法

* 可以使用dispatch\_queue\_create来创建对象，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；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，DISPATCH\_QUEUE\_CONCURRENT表示并发队列。
* // 串行队列的创建方法
* dispatch\_queue\_t queue= dispatch\_queue\_create("test.queue", DISPATCH\_QUEUE\_SERIAL);
* // 并发队列的创建方法
* dispatch\_queue\_t queue= dispatch\_queue\_create("test.queue", DISPATCH\_QUEUE\_CONCURRENT);
* 对于并发队列，还可以使用dispatch\_get\_global\_queue来创建**全局并发队列**。GCD默认提供了全局的并发队列，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。

### 2. 任务的创建方法

// 同步执行任务创建方法

dispatch\_sync(queue, ^{

NSLog(@"%@",[NSThread currentThread]); // 这里放任务代码

});

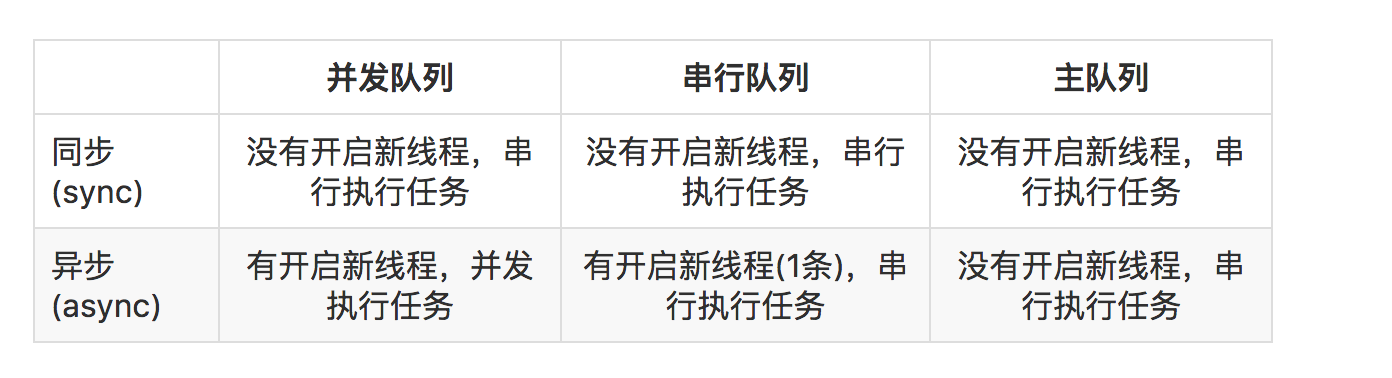
// 异步执行任务创建方法

dispatch\_async(queue, ^{

NSLog(@"%@",[NSThread currentThread]); // 这里放任务代码

});

虽然使用GCD只需两步，但是既然我们有两种队列，两种任务执行方式，那么我们就有了四种不同的组合方式。四种不同的组合方式是



#### 1. 并发队列 + 同步执行

* 不会开启新线程，执行完一个任务，再执行下一个任务

#### 2. 并发队列 + 异步执行

* 可同时开启多线程，任务交替执行

#### 3. 串行队列 + 同步执行

* 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. 串行队列 + 异步执行

* 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

主队列：GCD自带的一种特殊的**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
* 可使用dispatch\_get\_main\_queue()获得主队列

#### 5. 主队列 + 同步执行

* 互等卡住不可行(在主线程中调用)

这是因为我们在主线程中执行这段代码。我们把任务放到了主队列中，也就是放到了主线程的队列中。而同步执行有个特点，就是对于任务是立马执行的。那么当我们把第一个任务放进主队列中，它就会立马执行。但是主线程现在正在处理syncMain方法，所以任务需要等syncMain执行完才能执行。而syncMain执行到第一个任务的时候，又要等第一个任务执行完才能往下执行第二个和第三个任务。

那么，现在的情况就是syncMain方法和第一个任务都在等对方执行完毕。这样大家互相等待，所以就卡住了，所以我们的任务执行不了，而且syncMain---end也没有打印。

**要是如果不再主线程中调用，而在其他线程中调用会如何呢？**

* 不会开启新线程，执行完一个任务，再执行下一个任务（在其他线程中调用）
* 在其他线程中使用主队列 + 同步执行可看到：所有任务都是在主线程中执行的，并没有开启新的线程。而且由于主队列是串行队列，所以按顺序一个一个执行。
* 同时我们还可以看到，所有任务都在打印的syncConcurrent---begin和syncConcurrent---end之间，这说明任务是添加到队列中马上执行的。

#### 6. 主队列 + 异步执行

* 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线程之间的通讯

在iOS开发过程中，我们一般在主线程里边进行UI刷新，例如：点击、滚动、拖拽等事件。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。

### 1. GCD的栅栏方法 dispatch\_barrier\_async

* 我们有时需要异步执行两组操作，而且第一组操作执行完之后，才能开始执行第二组操作。这样我们就需要一个相当于栅栏一样的一个方法将两组异步执行的操作组给分割起来，当然这里的操作组里可以包含一个或多个任务。这就需要用到dispatch\_barrier\_async方法在两个操作组间形成栅栏。
* - (void)barrier
* {
* dispatch\_queue\_t queue = dispatch\_queue\_create("12312312", DISPATCH\_QUEUE\_CONCURRENT);
* dispatch\_async(queue, ^{
* NSLog(@"----1-----%@", [NSThread currentThread]);
* });
* dispatch\_async(queue, ^{
* NSLog(@"----2-----%@", [NSThread currentThread]);
* });
* dispatch\_barrier\_async(queue, ^{
* NSLog(@"----barrier-----%@", [NSThread currentThread]);
* });
* dispatch\_async(queue, ^{
* NSLog(@"----3-----%@", [NSThread currentThread]);
* });
* dispatch\_async(queue, ^{
* NSLog(@"----4-----%@", [NSThread currentThread]);
* });
* }

### 2. GCD的延时执行方法 dispatch\_after

* 当我们需要延迟执行一段代码时，就需要用到GCD的dispatch\_after方法。
* dispatch\_after(dispatch\_time(DISPATCH\_TIME\_NOW, (int64\_t)(2.0 \* NSEC\_PER\_SEC)), dispatch\_get\_main\_queue(), ^{
* // 2秒后异步执行这里的代码...
* NSLog(@"run-----");
* });

### 3. GCD的一次性代码(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方法。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。

static dispatch\_once\_t onceToken;

dispatch\_once(&onceToken, ^{

// 只执行1次的代码(这里面默认是线程安全的)

});

### 4. GCD的快速迭代方法 dispatch\_apply

* 通常我们会用for循环遍历，但是GCD给我们提供了快速迭代的方法dispatch\_apply，使我们可以同时遍历。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可以同时遍历多个数字。

dispatch\_queue\_t queue = dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0);

dispatch\_apply(6, queue, ^(size\_t index) {

NSLog(@"%zd------%@",index, [NSThread currentThread]);

});