iOS (http://lib.csdn.net/base/ios)

iOS (http://lib.csdn.net/base/ios) - 多线程 (http://lib.csdn.net/ios/node/675) - GCD (http://lib.csdn.net/ios/knowledge/1470)

iOS开发多线程-GCD介绍

作者：u011619223 (http://my.csdn.net/u011619223)

一、简单介绍

1.什么是GCD？

全称是Grand Central Dispatch，可译为“牛逼的中枢调度器”

纯C语言，提供了非常多强大的函数

2.GCD的优势

GCD是苹果公司为多核的并行运算提出的解决方案

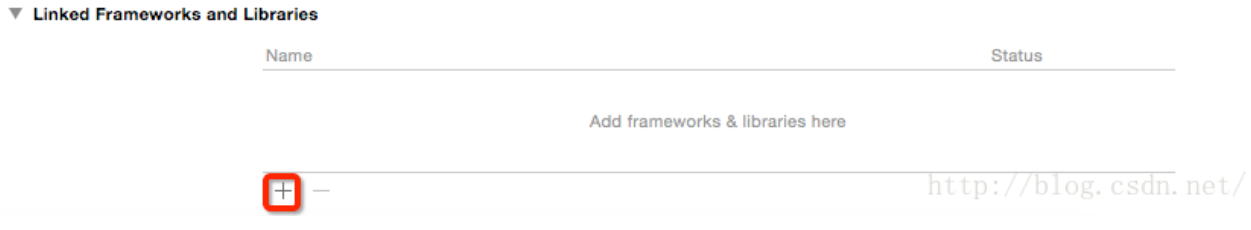
GCD会自动利用更多的CPU内核（比如双核、四核）

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）

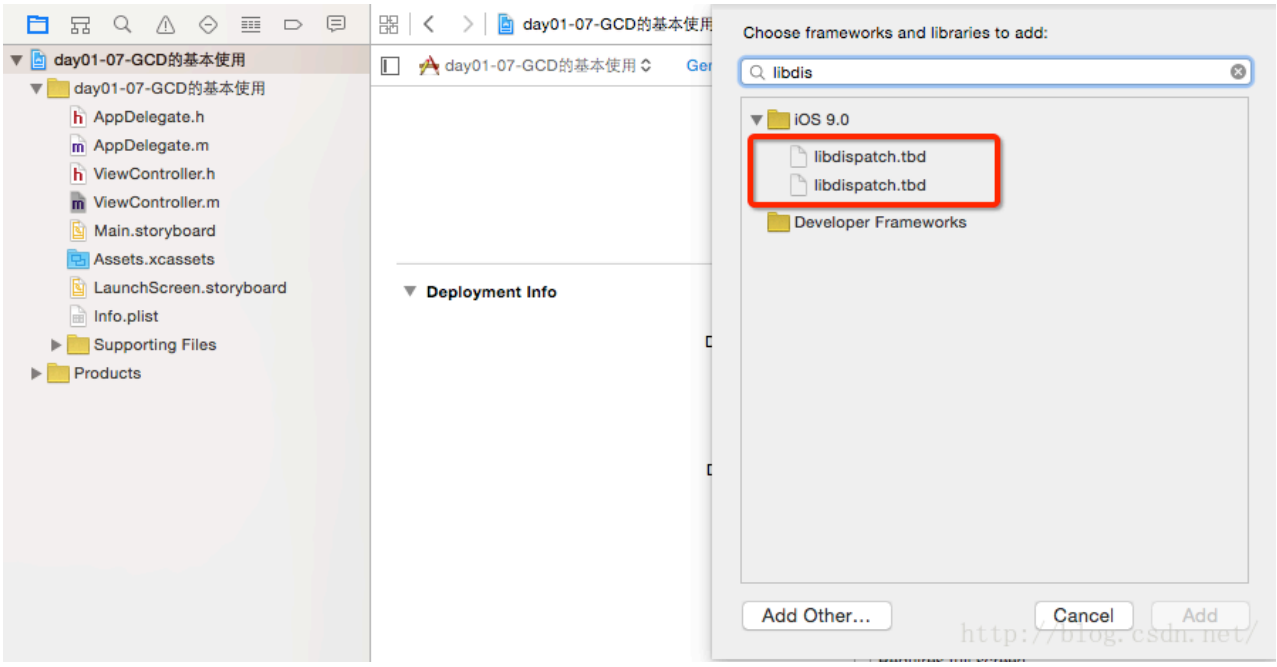
程序员只需要告诉GCD想要执行什么任务，不需要编写任何线程管理代码

3.提示

(1)GCD存在于libdispatch.dylib这个库中，这个调度库包含了GCD的所有的东西，但任何IOS程序，默认就加载了这个库，在程序运行的过程中会动态的加载这个库，不需要我们手动导入。



点击+a按钮，可以导入框架。



(2)GCD是纯C语言的，因此我们在编写GCD相关代码的时候，面对的函数，而不是方法。

(3)GCD中的函数大多数都以dispatch开头。

二、任务和队列

GCD中有2个核心概念

（1）任务：执行什么操作

（2）队列：用来存放任务

GCD的使用就2个步骤

（1）定制任务

（2）确定想做的事情

将任务添加到队列中，GCD会自动将队列中的任务取出，放到对应的线程中执行

提示：任务的取出遵循队列的FIFO原则：先进先出，后进后出

三、执行任务

1.GCD中有2个用来执行任务的函数

说明：把右边的参数（任务）提交给左边的参数（队列）进行执行。

（1）用同步的方式执行任务

```
dispatch_sync(dispatch_queue_t queue, dispatch_block_tblock);
```

参数说明：

queue：队列

block：任务

（2）用异步的方式执行任务

```
dispatch_async(dispatch_queue_t queue, dispatch_block_tblock);
```

2.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

3.GCD中还有个用来执行任务的函数

```
dispatch_barrier_async(dispatch_queue_t queue, dispatch_block_t block);
```

在前面的任务执行结束后它才执行,而且它后面的任务等它执行完成之后才会执行

这个queue不能是全局的并发队列

四、队列

1.队列的类型

GCD的队列可以分为2大类型

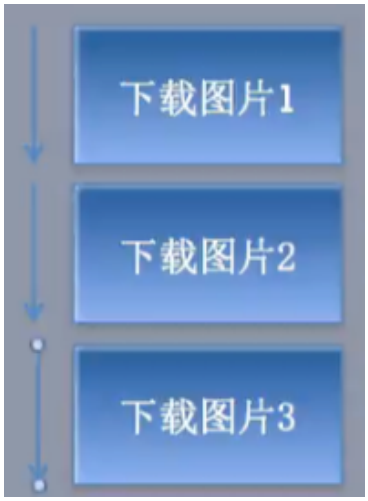
（1）并发队列（Concurrent Dispatch Queue）

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

（2）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）



2.补充说明

有4个术语比较容易混淆：同步、异步、并发、串行

同步和异步主要影响: 能不能开启新的线程

同步：只是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

并发和串行决定了任务的执行方式

并发：允许多个任务并发（同时）执行

串行：一个任务执行完毕后，再执行下一个任务

3.串行队列

GCD中获得串行有2种途径

（1）使用dispatch_queue_create函数创建串行队列

```
dispatch_queue_t  dispatch_queue_create(const char *label,  dispatch_queue_attr_t attr);
// 队列名称，  队列属性，一般用NULL即可
```

示例：

```
    // 创建串行队列（队列类型传递NULL或者DISPATCH_QUEUE_SERIAL）
    dispatch_queue_t queue = dispatch_queue_create("coderYLiu", NULL); // 创建
    dispatch_release(queue); // 非ARC需要释放手动创建的队列
```

（2）使用主队列（跟主线程相关联的队列）

主队列是GCD自带的一种特殊的串行队列,放在主队列中的任务，都会放到主线程中执行

使用dispatch_get_main_queue()获得主队列

示例：

```
dispatch_queue_t queue = dispatch_get_main_queue();
```

4.并发队列

GCD默认已经提供了全局的并发队列，供整个应用使用，不需要手动创建

使用dispatch_get_global_queue函数获得全局的并发队列

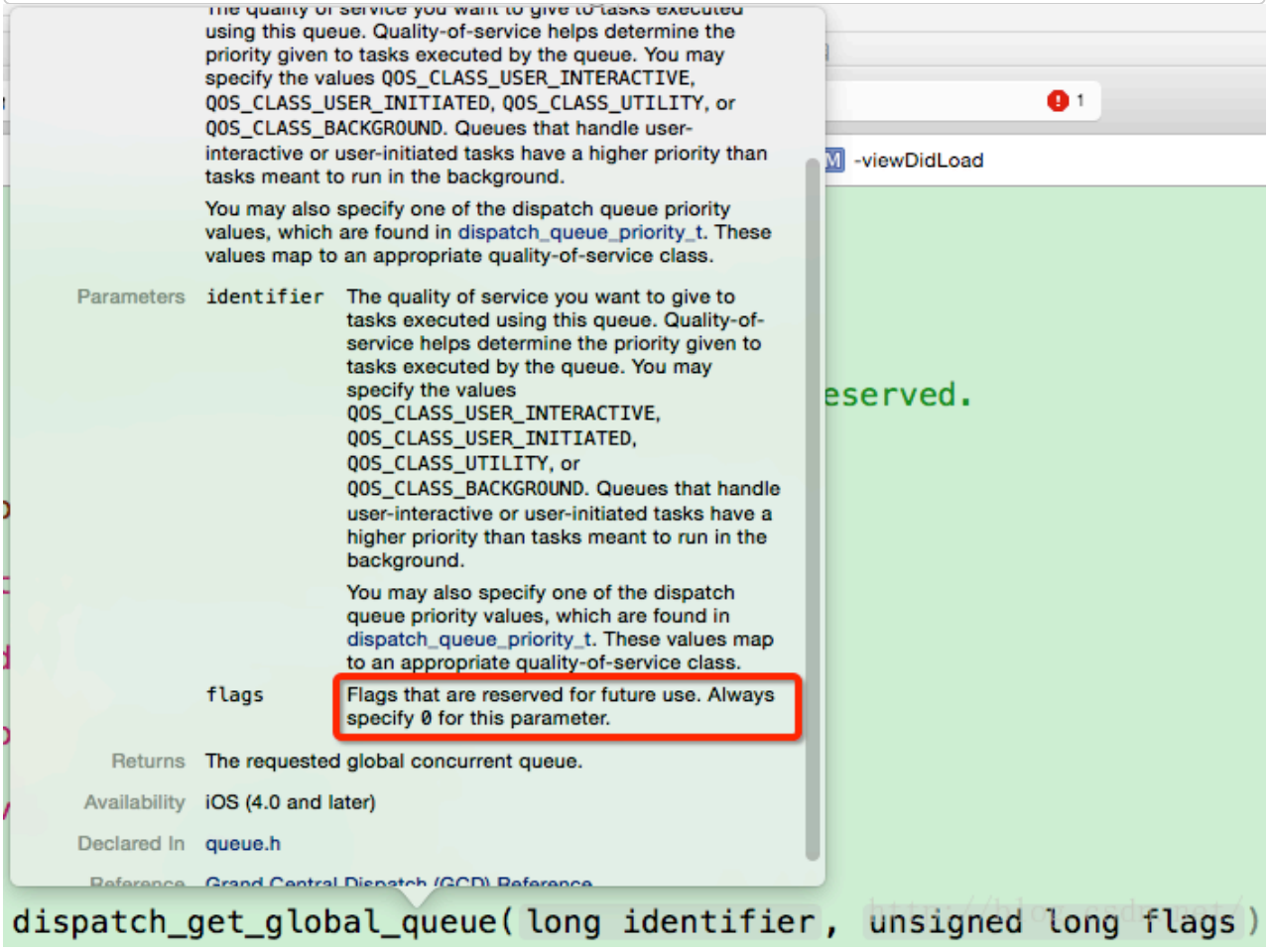
```
dispatch_queue_t dispatch_get_global_queue(dispatch_queue_priority_t priority,unsigned long flags); // 此参数暂时无用，用0即可
```

示例：

这个参数是留给以后用的，暂时用不上，传个0。

第一个参数为优先级，这里选择默认的。获取一个全局的默认优先级的并发队列。

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0); // 获得全局并发队列
```



说明：全局并发队列的优先级

```
#define DISPATCH_QUEUE_PRIORITY_HIGH 2 // 高

#define DISPATCH_QUEUE_PRIORITY_DEFAULT 0 // 默认（中）

#define DISPATCH_QUEUE_PRIORITY_LOW (-2) // 低

#define DISPATCH_QUEUE_PRIORITY_BACKGROUND INT16_MIN // 后台
```

5.各种队列的执行效果

	全局并发队列	手动创建串行队列	主队列
同步（sync）	p 没有开启新线程 p 串行执行任务	p 没有开启新线程 p 串行执行任务	p 没有开启新线程 p 串行执行任务
异步（async）	p 有开启新线程 p 并发执行任务	p 有开启新线程 p 串行执行任务	p 没有开启新线程 p 串行执行任务

注意:使用sync函数往当前串行队列中添加任务,会卡住当前的串行队列

五、代码示例

（1）用异步函数往并发队列中添加任务

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // 获取全局的并发队列
    dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0
);
    // 添加任务到队列中,就可以执行任务
    // 异步函数:具备开启新线程的能力
    /**
     *  参数1:队列
     *  参数2:封装任务的block
     */
    dispatch_async(queue, ^{
        NSLog(@"任务1-----%@", [NSThread currentThread]);
    });

    dispatch_async(queue, ^{
        NSLog(@"任务2-----%@", [NSThread currentThread]);
    });

    dispatch_async(queue, ^{
        NSLog(@"任务3-----%@", [NSThread currentThread]);
    });

    // 打印主线程
    NSLog(@"主线程-----%@", [NSThread mainThread]);

}

@end
```

```
2015-11-25 00:48:53.111 day01-07-GCD的基本使用[6261:113528] 主线程-----<NSThread: 0x7fc4230074b0>{number = 1, name = main}
2015-11-25 00:48:53.111 day01-07-GCD的基本使用[6261:113604] 任务1-----<NSThread: 0x7fc42305a240>{number = 2, name = (null)}
2015-11-25 00:48:53.111 day01-07-GCD的基本使用[6261:113608] 任务2-----<NSThread: 0x7fc422d1ae10>{number = 3, name = (null)}
2015-11-25 00:48:53.111 day01-07-GCD的基本使用[6261:113615] 任务3-----<NSThread: 0x7fc422f4e450>{number = 4, name = (null)}
```

总结：会开子线程,多条,队列中的任务并发执行

（2）用异步函数往串行队列中添加任务

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // 打印主线程
    NSLog(@"主线程-----%@", [NSThread mainThread]);

    /**
     *  创建串行队列
     *
     *  @param "CoderYLiu"            C语言的字符串 标签,名称
     *  @param DISPATCH_QUEUE_SERIAL 封装任务的block
     *
     *  DISPATCH_QUEUE_SERIAL       :串行
     *  DISPATCH_QUEUE_Concurrent   :并发
     */
    dispatch_queue_t queue = dispatch_queue_create("CoderYLiu", DISPATCH_QUEUE_SERIAL);

    // 添加任务到队列中执行
    dispatch_async(queue, ^{
        NSLog(@"任务1-----%@", [NSThread currentThread]);
    });

    dispatch_async(queue, ^{
        NSLog(@"任务2-----%@", [NSThread currentThread]);
    });

    dispatch_async(queue, ^{
        NSLog(@"任务3-----%@", [NSThread currentThread]);
    });

}

@end
```

```
2015-11-25 01:01:39.153 day01-07-GCD的基本使用[6290:120074] 主线程-----<NSThread: 0x7fb08b5002e0>{number = 1, name = main}
2015-11-25 01:01:39.154 day01-07-GCD的基本使用[6290:120144] 任务1-----<NSThread: 0x7fb08d1040e0>{number = 2, name = (null)}
2015-11-25 01:01:39.154 day01-07-GCD的基本使用[6290:120144] 任务2-----<NSThread: 0x7fb08d1040e0>{number = 2, name = (null)}
2015-11-25 01:01:39.155 day01-07-GCD的基本使用[6290:120144] 任务3-----<NSThread: 0x7fb08d1040e0>{number = 2, name = (null)}
```

总结：会开启线程，但是只开一条线程,队列中的任务串行执行

（3）用同步函数往并发队列中添加任务

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController
/**
 *  用同步函数往并发队列中添加任务
 */
- (void)viewDidLoad {
    [super viewDidLoad];
    // 打印主线程
    NSLog(@"主线程-----%@", [NSThread mainThread]);

    // 创建线程
    dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0
);

    // 添加任务到队列中执行
    dispatch_sync(queue, ^{
        NSLog(@"任务1-----%@", [NSThread currentThread]);
    });

    dispatch_sync(queue, ^{
        NSLog(@"任务2-----%@", [NSThread currentThread]);
    });

    dispatch_sync(queue, ^{
        NSLog(@"任务3-----%@", [NSThread currentThread]);
    });

}

@end
```

```
2015-11-25 01:14:50.270 day01-07-GCD的基本使用[6318:126765] 主线程-----<NSThread: 0x7fbce8c07fb0>{number = 1, name = main}
2015-11-25 01:14:50.270 day01-07-GCD的基本使用[6318:126765] 任务1-----<NSThread: 0x7fbce8c07fb0>{number = 1, name = main}
2015-11-25 01:14:50.270 day01-07-GCD的基本使用[6318:126765] 任务2-----<NSThread: 0x7fbce8c07fb0>{number = 1, name = main}
2015-11-25 01:14:50.271 day01-07-GCD的基本使用[6318:126765] 任务3-----<NSThread: 0x7fbce8c07fb0>{number = 1, name = main}
```

总结：不会开启新的线程，并发队列失去了并发的功能,串行执行

（4）用同步函数往串行队列中添加任务

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController
/**
 *  用同步函数往串行队列中添加任务
 */
- (void)viewDidLoad {
    [super viewDidLoad];

    NSLog(@"用同步函数往串行队列中添加任务");
    // 打印主线程
    NSLog(@"主线程-----%@", [NSThread mainThread]);

    // 创建串行队列
    dispatch_queue_t queue = dispatch_queue_create("CoderYLiu", DISPATCH_QUEUE_SERIAL);

    // 添加任务到队列中执行
    dispatch_sync(queue, ^{
        NSLog(@"任务1-----%@", [NSThread currentThread]);
    });

    dispatch_sync(queue, ^{
        NSLog(@"任务2-----%@", [NSThread currentThread]);
    });

    dispatch_sync(queue, ^{
        NSLog(@"任务3-----%@", [NSThread currentThread]);
    });
}

@end
```

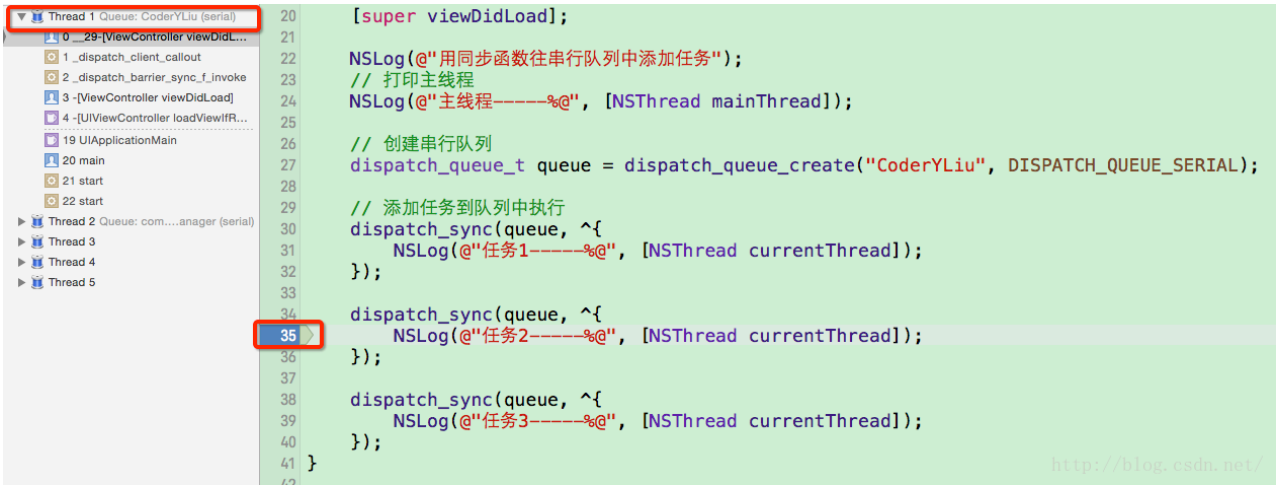
```
2015-11-25 01:22:17.833 day01-07-GCD的基本使用[6346:131153] 用同步函数往串行队列中添加任务
2015-11-25 01:22:17.833 day01-07-GCD的基本使用[6346:131153] 主线程-----<NSThread: 0x7fe6e250a5a0>{number = 1, name = main}
2015-11-25 01:22:17.833 day01-07-GCD的基本使用[6346:131153] 任务1-----<NSThread: 0x7fe6e250a5a0>{number = 1, name = main}
2015-11-25 01:22:17.834 day01-07-GCD的基本使用[6346:131153] 任务2-----<NSThread: 0x7fe6e250a5a0>{number = 1, name = main}
2015-11-25 01:22:17.834 day01-07-GCD的基本使用[6346:131153] 任务3-----<NSThread: 0x7fe6e250a5a0>{number = 1, name = main}
```

总结：不会开启新的线程,串行执行

（5）补充

补充：队列名称的作用：

将来调试的时候，可以看得出任务是在哪个队列中执行的。



（6）小结

说明：同步函数不具备开启线程的能力，无论是什么队列都不会开启线程；异步函数具备开启线程的能力，开启几条线程由队列决定（串行队列只会开启一条新的线程，并发队列会开启多条线程）。

同步函数

（1）并发队列：不会开线程

（2）串行队列：不会开线程

异步函数

（1）并发队列：能开启N条线程

（2）串行队列：开启1条线程

补充：

凡是函数中，各种函数名中带有create\copy\new\retain等字眼，都需要在不需要使用这个数据的时候进行release。

GCD的数据类型在ARC的环境下不需要再做release。

CF（core Foundation）的数据类型在ARC环境下还是需要做release。

异步函数具备开线程的能力，但不一定会开线程

查看原文>> (http://blog.csdn.net/u011619223/article/details/50021851)

0

看过本文的人也看了：

- iOS知识结构图 (http://lib.csdn.net/base/ios/structure)
- iOS中GCD的使用小结 (http://lib.csdn.net/article/ios/35924)
- iOS-GCD的学习记录(3) (http://lib.csdn.net/article/ios/38033)
- iOS笔记-- 多线程应该知道的那几件事... (http://lib.csdn.net/article/ios/39835)
- 【精】iOS GCD 详解 (http://lib.csdn.net/article/ios/40020)
- iOS_多线程_GCD (http://lib.csdn.net/article/ios/37109)

发表评论

输入评论内容

发表

0个评论