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多线程之GCD（3）

作者：tianzhipo5929 (http://my.csdn.net/tianzhipo5929)

一、延迟执行

1.介绍

iOS常见的延时执行有2种方式

（1）调用NSObject的方法

```
[self performSelector:@selector(run) withObject:nil afterDelay:2.0];

// 2秒后再调用self的run方法
```

（2）使用GCD函数

```
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...

});
```

2.说明

第一种方法，该方法在那个线程调用，那么run就在哪个线程执行（当前线程），通常是主线程。

```
[self performSelector:@selector(run) withObject:nil afterDelay:3.0];
```

说明：在3秒钟之后，执行run函数

代码示例：

```objc
//
//  YYViewController.m
//  01-GCD的常见使用（延迟执行）
//
//  Created by apple on 14-6-25.
//  Copyright (c) 2014年 itcase. All rights reserved.
//

#import "YYViewController.h"

@interface YYViewController ()

@end

@implementation YYViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    NSLog(@"打印线程----%@",[NSThread currentThread]);
    //延迟执行
    //第一种方法：延迟3秒钟调用run函数
    [self performSelector:@selector(run) withObject:nil afterDelay:2.0];

}
-(void)run
{
    NSLog(@"延迟执行----%@",[NSThread currentThread]);
}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    //在异步函数中执行
    dispatch_queue_t queue = dispatch_queue_create("wendingding", 0);

    dispatch_sync(queue, ^{
        [self performSelector:@selector(test) withObject:nil afterDelay:1.0];
    });
    NSLog(@"异步函数");
}
-(void)test
{
    NSLog(@"异步函数中延迟执行----%@",[NSThread currentThread]);
}
@end
```

说明：如果把该方法放在异步函数中执行，则方法不会被调用（BUG?）

```
2014-06-25 11:25:25.305 01-GCD的常见使用（延迟执行）[2224:60b] Cannot find executable for CFBundle 0x8c6a750 </Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator7.1.sdk/System/Library/AccessibilityBundles/CertUIFramework.axbundle> (not loaded)
2014-06-25 11:25:25.352 01-GCD的常见使用（延迟执行）[2224:60b] 打印线程----<NSThread: 0x8d4c6d0>{name = (null), num = 1}
2014-06-25 11:25:27.354 01-GCD的常见使用（延迟执行）[2224:60b] 延迟执行----<NSThread: 0x8d4c6d0>{name = (null), num = 1}
2014-06-25 11:25:32.670 01-GCD的常见使用（延迟执行）[2224:60b] 异步函数
2014-06-25 11:25:33.671 01-GCD的常见使用（延迟执行）[2224:60b] 异步函数中延迟执行----<NSThread: 0x8d4c6d0>{name = (null), num = 1}
```

第二种方法，

```
dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)
(5.0 * NSEC_PER_SEC)), dispatch_get_main_queue(), ^{

    //延迟执行的方法

});
```

说明：在5秒钟之后，执行block中的代码段。

参数说明：

```
#ifdef __BLOCKS__
__OSX_AVAILABLE_STARTING(__MAC_10_6,__IPHONE_4_0)
DISPATCH_EXPORT DISPATCH_NONNULL2 DISPATCH_NONNULL3 DISPATCH_NOTHROW
void
dispatch_after(dispatch_time_t when,
  dispatch_queue_t queue,
  dispatch_block_t block);
#endif
```

什么时间，执行这个队列中的这个任务。

代码示例：

```objc
//
//  YYViewController.m
//  02-GCD常见使用（延迟执行2）
//
//  Created by apple on 14-6-25.
//  Copyright (c) 2014年 itcase. All rights reserved.
//

#import "YYViewController.h"

@interface YYViewController ()

@end

@implementation YYViewController

- (void)viewDidLoad
{
    [super viewDidLoad];

    NSLog(@"打印当前线程---%@",  [NSThread currentThread]);

    //延迟执行，第二种方式
     //可以安排其线程(1),主队列
     dispatch_queue_t queue= dispatch_get_main_queue();
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(5.0 * NSEC_PER_SEC)), queue, ^{
        NSLog(@"主队列--延迟执行------%@",[NSThread currentThread]);
    });

    //可以安排其线程(2),并发队列
    //1.获取全局并发队列
    dispatch_queue_t queue1= dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
    //2.计算任务执行的时间
    dispatch_time_t when=dispatch_time(DISPATCH_TIME_NOW, (int64_t)(5.0 * NSEC_PER_SEC));
    //3.会在when这个时间点，执行queue中的这个任务
    dispatch_after(when, queue1, ^{
        NSLog(@"并发队列-延迟执行------%@",[NSThread currentThread]);
    });
}

@end
```

```
2014-06-25 11:28:04.972 02-GCD常见使用（延迟执行2）[2281:60b] Cannot find executable for CFBundle 0xaa5eb40 </Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator7.1.sdk/System/Library/AccessibilityBundles/CertUIFramework.axbundle> (not loaded)
2014-06-25 11:28:05.022 02-GCD常见使用（延迟执行2）[2281:60b] 打印当前线程---<NSThread: 0x8d24680>{name = (null), num = 1}
2014-06-25 11:28:10.024 02-GCD常见使用（延迟执行2）[2281:60b] 主队列--延迟执行------<NSThread: 0x8d24680>{name = (null), num = 1}
2014-06-25 11:28:10.024 02-GCD常见使用（延迟执行2）[2281:1303] 并发队列-延迟执行------<NSThread: 0x8c26a30>{name = (null), num = 3}
```

延迟执行：不需要再写方法，且它还传递了一个队列，我们可以指定并安排其线程。

如果队列是主队列，那么就在主线程执行，如果队列是并发队列，那么会新开启一个线程，在子线程中执行。

二、一次性代码

1.实现一次性代码

需求：点击控制器只有第一次点击的时候才打印。

实现代码：

```
//
//  YYViewController.m
//  03-GCD常见使用（一次性代码）
//
//  Created by apple on 14-6-25.
//  Copyright (c) 2014年 itcase. All rights reserved.
//

#import "YYViewController.h"

@interface YYViewController ()
@property(nonatomic,assign) BOOL log;
@end

@implementation YYViewController

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (_log==NO) {
        NSLog(@"该行代码只执行一次");
        _log=YES;
    }
}
@end
```

缺点：这是一个对象方法，如果又创建一个新的控制器，那么打印代码又会执行，因为每个新创建的控制器都有自己的布尔类型，且新创建的默认为NO，因此不能保证改行代码在整个程序中只打印一次。

2.使用dispatch_once一次性代码

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)

});
```

整个程序运行过程中，只会执行一次。

代码示例：

```
//
//  YYViewController.m
//  03-GCD常见使用（一次性代码）
//
//  Created by apple on 14-6-25.
//  Copyright (c) 2014年 itcase. All rights reserved.
//

#import "YYViewController.h"

@interface YYViewController ()
@property(nonatomic,assign) BOOL log;
@end

@implementation YYViewController

//-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
//{
//    if (_log==NO) {
//        NSLog(@"该行代码只执行一次");
//        _log=YES;
//    }
//}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        NSLog(@"该行代码只执行一次");
    });
}
@end
```

效果（程序运行过程中，打印代码只会执行一次）：

```
2014-06-25 11:30:05.433 03-GCD常见使用（一次性代码）[2331:60b] Cannot find executable for CFBundle 0x8e3e570 </Applications/Xcode.app/
Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator7.1.sdk/System/Library/AccessibilityBundles/
CertUIFramework.axbundle> (not loaded)
2014-06-25 11:30:06.308 03-GCD常见使用（一次性代码）[2331:60b] 该行代码只执行一次
```

三、队列组

需求：从网络上下载两张图片，把两张图片合并成一张最终显示在view上。

1.第一种方法

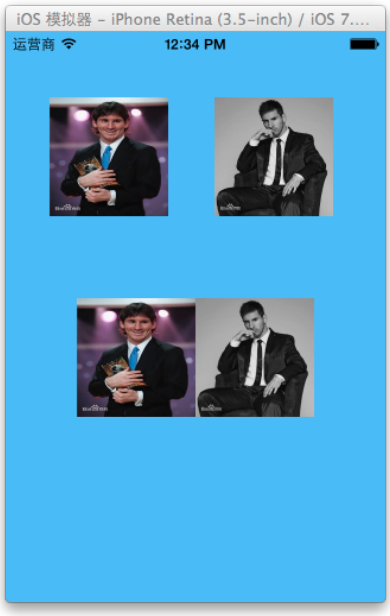
代码示例：

```
//
//  YYViewController.m
```

```
 3 //  04-GCD基本使用（队列组下载图片）
 4 //
 5 //  Created by apple on 14-6-25.
 6 //  Copyright (c) 2014年 itcase. All rights reserved.
 7 //
 8
 9 #import "YYViewController.h"
10 //宏定义全局并发队列
11 #define global_quque    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEF
AULT, 0)
12 //宏定义主队列
13 #define main_queue       dispatch_get_main_queue()
14
15 @interface YYViewController ()
16 @property (weak, nonatomic) IBOutlet UIImageView *imageView1;
17 @property (weak, nonatomic) IBOutlet UIImageView *imageView2;
18 @property (weak, nonatomic) IBOutlet UIImageView *imageView3;
19
20 @end
21
22 @implementation YYViewController
23
24 - (void)viewDidLoad
25 {
26     [super viewDidLoad];
27 }
28 -(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
29 {
30     //获取全局并发队列
31 //    dispatch_queue_t queue= dispatch_get_global_queue(DISPATCH_QUEUE_PRIORI
TY_DEFAULT, 0);
32     //获取主队列
33 //    dispatch_queue_t queue= dispatch_get_main_queue();
34
35 //    图片1: http://d.hiphotos.baidu.com/baike/c0%3Dbaike80%2C5%2C5%2C80%2C26/
sign=2b9a12172df5e0fefa1581533d095fcd/cefc1e178a82b9019115de3d738da9773912ef00.j
pg
36 //    图片2: http://h.hiphotos.baidu.com/baike/c0%3Dbaike80%2C5%2C5%2C80%2C26/
sign=f47fd63ca41ea8d39e2f7c56f6635b2b/1e30e924b899a9018b8d3ab11f950a7b0308f5f9.j
pg
37     dispatch_async(global_quque, ^{
38         //下载图片1
39        UIImage *image1= [self imageWithUrl:@"http://d.hiphotos.baidu.com/baik
e/c0%3Dbaike80%2C5%2C5%2C80%2C26/sign=2b9a12172df5e0fefa1581533d095fcd/cefc1e178
a82b9019115de3d738da9773912ef00.jpg"];
40         NSLog(@"图片1下载完成---%@",[NSThread currentThread]);
41
42         //下载图片2
43        UIImage *image2= [self imageWithUrl:@"http://h.hiphotos.baidu.com/baik
e/c0%3Dbaike80%2C5%2C5%2C80%2C26/sign=f47fd63ca41ea8d39e2f7c56f6635b2b/1e30e924b
899a9018b8d3ab11f950a7b0308f5f9.jpg"];
44         NSLog(@"图片2下载完成---%@",[NSThread currentThread]);
45
46         //回到主线程显示图片
47         dispatch_async(main_queue, ^{
48              NSLog(@"显示图片---%@",[NSThread currentThread]);
```

```
49              self.imageView1.image=image1;
50              self.imageView2.image=image2;
51              //合并两张图片
52              UIGraphicsBeginImageContextWithOptions(CGSizeMake(200, 100), NO,
0.0);
53              [image1 drawInRect:CGRectMake(0, 0, 100, 100)];
54              [image2 drawInRect:CGRectMake(100, 0, 100, 100)];
55              self.imageView3.image=UIGraphicsGetImageFromCurrentImageContext()
;
56              //关闭上下文
57              UIGraphicsEndImageContext();
58                 NSLog(@"图片合并完成---%@",[NSThread currentThread]);
59          });
60          //
61      });
62 }
63
64 //封装一个方法，传入一个url参数，返回一张网络上下载的图片
65 -(UIImage *)imageWithUrl:(NSString *)urlStr
66 {
67      NSURL *url=[NSURL URLWithString:urlStr];
68      NSData *data=[NSData dataWithContentsOfURL:url];
69      UIImage *image=[UIImage imageWithData:data];
70      return image;
71 }
72 @end
```

显示效果：



打印查看：

```
2014-06-25 12:33:44.253 04-GCD基本使用（队列组下载图片）[1625:1303] 图片1下载完成---<NSThread: 0x8dc9380>{name = (null), num = 3}
2014-06-25 12:33:44.340 04-GCD基本使用（队列组下载图片）[1625:1303] 图片2下载完成---<NSThread: 0x8dc9380>{name = (null), num = 3}
2014-06-25 12:33:44.341 04-GCD基本使用（队列组下载图片）[1625:70b] 显示图片---<NSThread: 0x8a90780>{name = (null), num = 1}
2014-06-25 12:33:44.391 04-GCD基本使用（队列组下载图片）[1625:70b] 图片合并完成---<NSThread: 0x8a90780>{name = (null), num = 1}
```

问题：这种方式的效率不高，需要等到图片1.图片2都下载完成后才行。

提示：使用队列组可以让图片1和图片2的下载任务同时进行，且当两个下载任务都完成的时候回到主线程进行显示。

2.使用队列组解决

步骤：

创建一个组

开启一个任务下载图片1

开启一个任务下载图片2

同时执行下载图片1\下载图片2操作

等group中的所有任务都执行完毕, 再回到主线程执行其他操作

代码示例

```
 1 //
 2 //  YYViewController.m
 3 //  04-GCD基本使用（队列组下载图片）
 4 //
 5 //  Created by apple on 14-6-25.
 6 //  Copyright (c) 2014年 itcase. All rights reserved.
 7 //
 8
 9 #import "YYViewController.h"
10 //宏定义全局并发队列
11 #define global_quque    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DE
FAULT, 0)
12 //宏定义主队列
13 #define main_queue       dispatch_get_main_queue()
14
15 @interface YYViewController ()
16 @property (weak, nonatomic) IBOutlet UIImageView *imageView1;
17 @property (weak, nonatomic) IBOutlet UIImageView *imageView2;
18 @property (weak, nonatomic) IBOutlet UIImageView *imageView3;
19
20 @end
21
22 @implementation YYViewController
23
24 - (void)viewDidLoad
25 {
26     [super viewDidLoad];
27 }
28 -(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
29 {
30     //    图片1: http://d.hiphotos.baidu.com/baike/c0%3Dbaike80%2C5%2C5%2C80%
2C26/sign=2b9a12172df5e0fefa1581533d095fcd/cefc1e178a82b9019115de3d738da9773912e
f00.jpg
```

```
31     //    图片2：http://h.hiphotos.baidu.com/baike/c0%3Dbaike80%2C5%2C5%2C80%2C26/sign=f47fd63ca41ea8d39e2f7c56f6635b2b/1e30e924b899a9018b8d3ab11f950a7b0308f5f9.jpg
32
33
34     //1.创建一个队列组
35         dispatch_group_t group = dispatch_group_create();
36
37     //2.开启一个任务下载图片1
38     __block UIImage *image1=nil;
39     dispatch_group_async(group, global_quque, ^{
40         image1= [self imageWithUrl:@"http://d.hiphotos.baidu.com/baike/c0%3Dbaike80%2C5%2C5%2C80%2C26/sign=2b9a12172df5e0fefa1581533d095fcd/cefc1e178a82b9019115de3d738da9773912ef00.jpg"];
41         NSLog(@"图片1下载完成---%@",[NSThread currentThread]);
42     });
43
44     //3.开启一个任务下载图片2
45     __block UIImage *image2=nil;
46     dispatch_group_async(group, global_quque, ^{
47         image2= [self imageWithUrl:@"http://h.hiphotos.baidu.com/baike/c0%3Dbaike80%2C5%2C5%2C80%2C26/sign=f47fd63ca41ea8d39e2f7c56f6635b2b/1e30e924b899a9018b8d3ab11f950a7b0308f5f9.jpg"];
48         NSLog(@"图片2下载完成---%@",[NSThread currentThread]);
49     });
50
51     //同时执行下载图片1\下载图片2操作
52
53    //4.等group中的所有任务都执行完毕，再回到主线程执行其他操作
54     dispatch_group_notify(group,main_queue, ^{
55         NSLog(@"显示图片---%@",[NSThread currentThread]);
56         self.imageView1.image=image1;
57         self.imageView2.image=image2;
58
59         //合并两张图片
60         //注意最后一个参数是浮点数（0.0），不要写成0。
61         UIGraphicsBeginImageContextWithOptions(CGSizeMake(200, 100), NO, 0.0);
62         [image1 drawInRect:CGRectMake(0, 0, 100, 100)];
63         [image2 drawInRect:CGRectMake(100, 0, 100, 100)];
64         self.imageView3.image=UIGraphicsGetImageFromCurrentImageContext();
65         //关闭上下文
66         UIGraphicsEndImageContext();
67
68         NSLog(@"图片合并完成---%@",[NSThread currentThread]);
69     });
70
71 }
72 -(void)download2image
73 {
74     //获取全局并发队列
75 //    dispatch_queue_t queue= dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
76     //获取主队列
77 //    dispatch_queue_t queue= dispatch_get_main_queue();
78
```

```
 79     dispatch_async(global_quque, ^{
 80         //下载图片1
 81        UIImage *image1= [self imageWithUrl:@"http://news.baidu.com/z/resourc
e/r/image/2014-06-22/2a1009253cf9fc7c97893a4f0fe3a7b1.jpg"];
 82         NSLog(@"图片1下载完成---%@",[NSThread currentThread]);
 83 
 84         //下载图片2
 85        UIImage *image2= [self imageWithUrl:@"http://news.baidu.com/z/resourc
e/r/image/2014-06-22/2a1009253cf9fc7c97893a4f0fe3a7b1.jpg"];
 86         NSLog(@"图片2下载完成---%@",[NSThread currentThread]);
 87 
 88         //回到主线程显示图片
 89         dispatch_async(main_queue, ^{
 90              NSLog(@"显示图片---%@",[NSThread currentThread]);
 91             self.imageView1.image=image1;
 92             self.imageView2.image=image2;
 93             //合并两张图片
 94             UIGraphicsBeginImageContextWithOptions(CGSizeMake(200, 100), NO,
0.0);
 95             [image1 drawInRect:CGRectMake(0, 0, 100, 100)];
 96             [image2 drawInRect:CGRectMake(0, 0, 100, 100)];
 97             self.imageView3.image=UIGraphicsGetImageFromCurrentImageContext(
);
 98             //关闭上下文
 99             UIGraphicsEndImageContext();
100                NSLog(@"图片合并完成---%@",[NSThread currentThread]);
101         });
102         //
103     });
104 }
105 
106 //封装一个方法，传入一个url参数，返回一张网络上下载的图片
107 -(UIImage *)imageWithUrl:(NSString *)urlStr
108 {
109     NSURL *url=[NSURL URLWithString:urlStr];
110     NSData *data=[NSData dataWithContentsOfURL:url];
111     UIImage *image=[UIImage imageWithData:data];
112     return image;
113 }
114 @end
```

打印查看（同时开启了两个子线程，分别下载图片）：

```
2014-06-25 12:38:04.395 04-GCD基本使用（队列组下载图片）[1720:1307] 图片2下载完成---<NSThread: 0x8a49da0>{name = (null), num = 3}
2014-06-25 12:38:04.405 04-GCD基本使用（队列组下载图片）[1720:313] 图片1下载完成---<NSThread: 0x8b5dd90>{name = (null), num = 4}
2014-06-25 12:38:04.405 04-GCD基本使用（队列组下载图片）[1720:70b] 显示图片---<NSThread: 0x8b58220>{name = (null), num = 1}
2014-06-25 12:38:04.415 04-GCD基本使用（队列组下载图片）[1720:70b] 图片合并完成---<NSThread: 0x8b58220>{name = (null), num = 1}
```

2.补充说明

有这么1种需求：

首先：分别异步执行2个耗时的操作

其次：等2个异步操作都执行完毕后，再回到主线程执行操作

如果想要快速高效地实现上述需求，可以考虑用队列组

```
dispatch_group_t group =  dispatch_group_create();

dispatch_group_async(group, dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFA
^{

   // 执行1个耗时的异步操作

});

dispatch_group_async(group, dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFA
^{

   // 执行1个耗时的异步操作

});

dispatch_group_notify(group, dispatch_get_main_queue(), ^{

   // 等前面的异步操作都执行完毕后，回到主线程...

});
```

本文转载自：http://www.cnblogs.com/wendingding/p/3807716.html

查看原文>> (http://blog.csdn.net/tianzhipo5929/article/details/43954341)

1

看过本文的人也看了：

- iOS知识结构图 (http://lib.csdn.net/base/ios/structure)
- iOS开发多线程-GCD介绍 (http://lib.csdn.net/article/ios/45415)
- iOS中GCD的使用小结 (http://lib.csdn.net/article/ios/35924)
- iOS笔记-- 多线程应该知道的那几件事... (http://lib.csdn.net/article/ios/39835)
- <iOS>GCD使用经验与技巧 (http://lib.csdn.net/article/ios/38792)
- iOS_多线程_GCD (http://lib.csdn.net/article/ios/37109)

发表评论

输入评论内容

发表

1个评论


(http://my.csdn.net/qq_29880265)

qq_29880265 (http://my.csdn.net/qq_29880265)

受教了，谢谢。

2016-09-27 11:45:18 回复