日志

返回日志列表

GCD中Group 和 串行队列（Serial Dispatch Queue） 2016-8-31 17:57

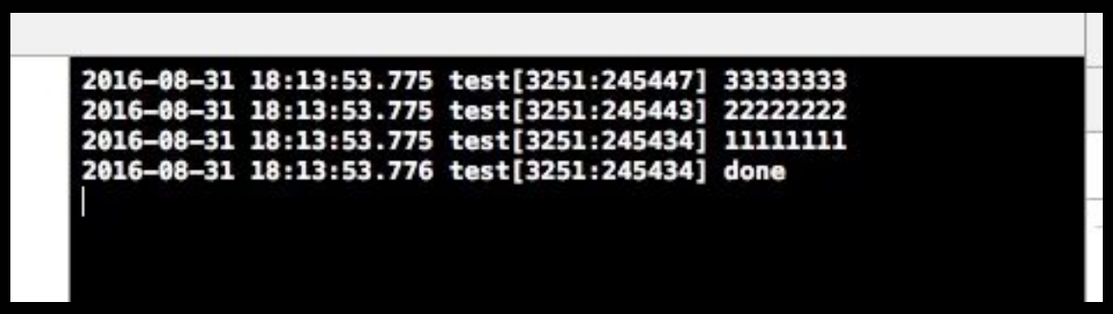
GCD中Group 和 串行队列（Serial Dispatch Queue）

Dispatch Group

在追加到Dispatch Queue中的多个任务处理完毕之后想执行结束处理，这种需求会经常出现。如果只是使用一个Serial Dispatch Queue（串行队列）时，只要将想执行的处理全部追加到该串行队列中并在最后追加结束处理即可。
但是在使用Concurrent Queue 时，可能会同时使用多个Dispatch Queue时，源代码就会变得很复杂。在这种情况下，就可以使用Dispatch Group。

```
    dispatch_group_t group = dispatch_group_create();
    dispatch_queue_t queue = dispatch_queue_create("http://user.qzone.qq.com/78269391",
DISPATCH_QUEUE_CONCURRENT);

    dispatch_group_async(group, queue, ^{
        for (int i = 0; i < 100000; i++) {
            if (i == 99999) {
                NSLog(@"11111111");
            }
        }

    });

    dispatch_group_async(group, queue, ^{
        NSLog(@"22222222");
    });

    dispatch_group_async(group, queue, ^{
        NSLog(@"33333333");
    });

    dispatch_group_notify(group, queue, ^{
        NSLog(@"done");
    });
```

控制台日志：



在使用Group时，因为向Concurrent Dispatch Queue 追加处理，多个线程并行执行，所以追加处理的执行顺序不定。执行顺序会发生变化，但是此执行结果的done一定是最后输出的。

无论向什么样的Dispatch Queue中追加处理，使用Dispatch Group都可以监视这些处理执行的结果。一旦检测到所有处理执行结束，就可以将结束的处理追加到Dispatch Queue中，这就是使用Dispatch Group的原因。

下面是一个使用Dispatch Group异步下载两张图片，然后合并成一张图片的demo（在主线程中更新UI）：

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    self.view.backgroundColor = [UIColor whiteColor];


    [self operation];


}


- (void)operation {
    UILabel *textLabel = [[UILabel alloc] initWithFrame:CGRectMake(KmainScreenWith/2 - 100, 50, 200, 40)];

    textLabel.textAlignment = NSTextAlignmentCenter;
    textLabel.textColor = [UIColor redColor];
    [self.view addSubview:textLabel];
    self.textLabel = textLabel;

    UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(KmainScreenWith/2 - 100, 100,
     200, 400)];
    [self.view addSubview:imageView];
    self.imageview = imageView;



    [self group];

//    [self group2];


    NSLog(@"在下载图片的时候，主线程貌似还可以干点什么");



}


- (void)group {//使用 Group 来执行任务组
    __block UIImage *image1 = [[UIImage alloc] init];
    __block UIImage *image2 = [[UIImage alloc] init];

    dispatch_group_t group = dispatch_group_create();
    dispatch_queue_t queue = dispatch_queue_create("78269391", DISPATCH_QUEUE_CONCURRENT);
    dispatch_group_async(group, queue, ^{

//        dispatch_async(queue, ^{
//            sleep(10);
//            NSLog(@"异步任务完成");
//        });
//
        sleep(10);
        NSLog(@"正在下载第一张图片");
        NSData *data = [NSData dataWithContentsOfURL:[NSURL
URLWithString:@"http://e.hiphotos.baidu.com/image/h%3D360/sign=be08bc32718da977512f802d8050f872/91529822720e
0cf3235231fa0846f21fbe09aa0e.jpg"]];
        NSLog(@"第一张图片下载完毕");
        image1 = [UIImage imageWithData:data];
    });

    dispatch_group_async(group, queue, ^{


        NSLog(@"正在下载第二张图片");
        NSData *data = [NSData dataWithContentsOfURL:[NSURL
URLWithString:@"http://h.hiphotos.baidu.com/image/h%3D360/sign=bad7972a0cf41bd5c553eef261db81a0/f9198618367a
dab48d246b4289d4b31c8701e43f.jpg"]];

        NSLog(@"第二张图片下载完毕");
```

```
    image2 = [UIImage imageWithData:data];
    });



    __weak typeof(self) weakSelf = self;
    dispatch_group_notify(group, queue, ^{
        __strong typeof(weakSelf) strongSelf = weakSelf;


        UIGraphicsBeginImageContext(strongSelf.imageview.bounds.size);

        [image1 drawInRect:CGRectMake(0, 0, CGRectGetWidth(strongSelf.imageview.bounds),
CGRectGetHeight(strongSelf.imageview.bounds)/2)];
        [image2 drawInRect:CGRectMake(0, CGRectGetHeight(strongSelf.imageview.bounds)/2,
CGRectGetWidth(strongSelf.imageview.bounds), CGRectGetHeight(strongSelf.imageview.bounds)/2)];

        UIImage *newImage = UIGraphicsGetImageFromCurrentImageContext();
        UIGraphicsEndImageContext();

        dispatch_async(dispatch_get_main_queue(), ^{


            NSLog(@"所有任务都处理完成");
            strongSelf.imageview.image = newImage;
            strongSelf.textLabel.text = @"图片合并完毕";
        });
    });
}

- (void)group2{
    __block UIImage *image1 = [[UIImage alloc] init];
    __block UIImage *image2 = [[UIImage alloc] init];

    dispatch_queue_t queue = dispatch_queue_create(0, NULL);

    dispatch_async(queue, ^{
        sleep(10);
        NSLog(@"正在下载第一张图片");
        NSData *data = [NSData dataWithContentsOfURL:[NSURL
URLWithString:@"http://e.hiphotos.baidu.com/image/h%3D360/sign=be08bc32718da977512f802d8050f872/91529822720e
0cf3235231fa0846f21fbe09aa0e.jpg"]];
        NSLog(@"第一张图片下载完毕");
        image1 = [UIImage imageWithData:data];
    });
    dispatch_async(queue, ^{
        NSLog(@"正在下载第二张图片");
        NSData *data = [NSData dataWithContentsOfURL:[NSURL
URLWithString:@"http://h.hiphotos.baidu.com/image/h%3D360/sign=bad7972a0cf41bd5c553eef261db81a0/f9198618367a
dab48d246b4289d4b31c8701e43f.jpg"]];

        NSLog(@"第二张图片下载完毕");
        image2 = [UIImage imageWithData:data];
    });


    __weak typeof(self) weakSelf = self;

    dispatch_async(queue, ^{

        __strong typeof(weakSelf) strongSelf = weakSelf;

        UIGraphicsBeginImageContext(strongSelf.imageview.bounds.size);

        [image1 drawInRect:CGRectMake(0, 0, CGRectGetWidth(strongSelf.imageview.bounds),
CGRectGetHeight(strongSelf.imageview.bounds)/2)];
        [image2 drawInRect:CGRectMake(0, CGRectGetHeight(strongSelf.imageview.bounds)/2,
CGRectGetWidth(strongSelf.imageview.bounds), CGRectGetHeight(strongSelf.imageview.bounds)/2)];

        UIImage *newImage = UIGraphicsGetImageFromCurrentImageContext();
        UIGraphicsEndImageContext();

        dispatch_async(dispatch_get_main_queue(), ^{

            NSLog(@"所有任务都处理完成");
            strongSelf.imageview.image = newImage;
            strongSelf.textLabel.text = @"图片合并完毕";
        });


    });
}
```

这是使用Group时的控制台输出 [self group];

```
2016-08-31 18:10:33.552 test[3194:240197] 在下载图片的时候，主线程貌
似还可以干点什么
2016-08-31 18:10:33.552 test[3194:240252] 正在下载第二张图片
2016-08-31 18:10:33.605 test[3194:240252] 第二张图片下载完毕
2016-08-31 18:10:43.556 test[3194:240385] 正在下载第一张图片
2016-08-31 18:10:43.599 test[3194:240385] 第一张图片下载完毕
2016-08-31 18:10:43.610 test[3194:240197] 所有任务都处理完成
```

这是使用串行队列的控制台输出 [self group2];

```
2016-08-31 18:11:34.958 test[3224:241916] 在下载图片的时候，主线程貌
似还可以干点什么
2016-08-31 18:11:44.959 test[3224:241954] 正在下载第一张图片
2016-08-31 18:11:45.023 test[3224:241954] 第一张图片下载完毕
2016-08-31 18:11:45.024 test[3224:241954] 正在下载第二张图片
2016-08-31 18:11:45.045 test[3224:241954] 第二张图片下载完毕
2016-08-31 18:11:45.056 test[3224:241916] 所有任务都处理完成
```

虽然两种方法都能实现需求，但是很明显在控制台输出的时间上可以看出来 Group 是并发执行多个任务，但只有所有任务都执行完之后才会去刷新UI；而串行队列则需要依据加入队列的顺序依次执行，上一个任务执行完才会执行下一个，这样就会影响时间以及效率。

效果图:

#pragma mark - 问题

问题：在Group 中 如果在异步任务中嵌入另一个异步任务 ，Group执行顺序会乱掉 （目前还没研究透彻，我再探究探究。。。。）

```objc
__block UIImage *image1 = [[UIImage alloc] init];
__block UIImage *image2 = [[UIImage alloc] init];

dispatch_group_t group = dispatch_group_create();
dispatch_queue_t queue = dispatch_queue_create("78269391", DISPATCH_QUEUE_CONCURRENT);
dispatch_group_async(group, queue, ^{

    dispatch_async(queue, ^{
        sleep(10);
        NSLog(@"异步任务完成");
    });
```

```
//        sleep(10);
//        NSLog(@"正在下载第一张图片");
//        NSData *data = [NSData dataWithContentsOfURL:[NSURL
URLWithString:@"http://e.hiphotos.baidu.com/image/h%3D360/sign=be08bc32718da977512f802d8050f872/91529822720e
0cf3235231fa0846f21fbe09aa0e.jpg"]];
//        NSLog(@"第一张图片下载完毕");
//        image1 = [UIImage imageWithData:data];
    });

    dispatch_group_async(group, queue, ^{


        NSLog(@"正在下载第二张图片");
        NSData *data = [NSData dataWithContentsOfURL:[NSURL
URLWithString:@"http://h.hiphotos.baidu.com/image/h%3D360/sign=bad7972a0cf41bd5c553eef261db81a0/f9198618367a
dab48d246b4289d4b31c8701e43f.jpg"]];

        NSLog(@"第二张图片下载完毕");
        image2 = [UIImage imageWithData:data];
    });



   __weak typeof(self) weakSelf = self;
    dispatch_group_notify(group, queue, ^{
        __strong typeof(weakSelf) strongSelf = weakSelf;


        UIGraphicsBeginImageContext(strongSelf.imageview.bounds.size);

        [image1 drawInRect:CGRectMake(0, 0, CGRectGetWidth(strongSelf.imageview.bounds),
CGRectGetHeight(strongSelf.imageview.bounds)/2)];
        [image2 drawInRect:CGRectMake(0, CGRectGetHeight(strongSelf.imageview.bounds)/2,
CGRectGetWidth(strongSelf.imageview.bounds), CGRectGetHeight(strongSelf.imageview.bounds)/2)];

        UIImage *newImage = UIGraphicsGetImageFromCurrentImageContext();
        UIGraphicsEndImageContext();

        dispatch_async(dispatch_get_main_queue(), ^{


            NSLog(@"所有任务都处理完成");
            strongSelf.imageview.image = newImage;
            strongSelf.textLabel.text = @"图片合并完毕";
        });
    });
```

```
2016-08-31 18:06:10.215 test[3148:235119] 在下载图片的时候，主线程貌
似还可以干点什么
2016-08-31 18:06:10.215 test[3148:235152] 正在下载第二张图片
2016-08-31 18:06:10.269 test[3148:235152] 第二张图片下载完毕
2016-08-31 18:06:10.278 test[3148:235119] 所有任务都处理完成
2016-08-31 18:06:20.218 test[3148:235158] 异步任务完成
```

接上面 已解决嵌套异步线程问题 使用
dispatch_group_enter 或者 dispatch_semaphore_t 信号量

```
//
//  ViewController.m
//  GCDtest
//
//  Created by Mac on 2016/12/29.
//  Copyright © 2016年 Mac. All rights reserved.
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.


    [self doEnter];
//    [self doSemaphore1];
//    [self doSemaphore2];
```

```objc
}


//enter方式
- (void)doEnter{
    dispatch_group_t group = dispatch_group_create();//创建group
    dispatch_queue_t queue = dispatch_queue_create(nil, DISPATCH_QUEUE_SERIAL);//串行队列
    dispatch_queue_t global_queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);//全局并行队列

 /*
  dispatch_group_enter 和 dispatch_group_leave 必须成对出现
  dispatch_group_enter 使group任务 +1 ;
  dispatch_group_leave 使group 任务 -1;
  */


    dispatch_group_enter(group);
    dispatch_group_async(group, queue, ^{
        //        NSLog(@"1");
        dispatch_async(global_queue, ^{
            sleep(8);
            NSLog(@"1");
            dispatch_group_leave(group);

        });


    });

    dispatch_group_enter(group);

    dispatch_group_async(group, queue, ^{
        //        NSLog(@"2");
        dispatch_async(global_queue, ^{
            sleep(5);
            NSLog(@"2");
            dispatch_group_leave(group);
        });

    });


    dispatch_group_enter(group);

    dispatch_group_async(group, queue, ^{
        //        NSLog(@"3");
        dispatch_async(global_queue, ^{
            NSLog(@"3");
            dispatch_group_leave(group);
        });
    });

    dispatch_group_enter(group);

    dispatch_group_async(group, queue, ^{
        //        sleep(5);
        //        NSLog(@"4");
        dispatch_async(global_queue, ^{
            NSLog(@"4");
            dispatch_group_leave(group);
        });
    });

    //    异步监听所有任务完成
    dispatch_group_notify(group, queue, ^{
        NSLog(@"all");
    });

    NSLog(@"主线程");


}

//信号量方式
- (void)doSemaphore1{
    dispatch_group_t group = dispatch_group_create();//创建group
    dispatch_queue_t queue = dispatch_queue_create(nil, DISPATCH_QUEUE_SERIAL);//串行队列
    dispatch_queue_t global_queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);//全局并行队列
    dispatch_group_async(group, queue, ^{
        //        NSLog(@"1");
        dispatch_semaphore_t semaphore = dispatch_semaphore_create(0);//创建一个信号量 值为初始值
        dispatch_async(global_queue, ^{
            sleep(6);
            NSLog(@"1");
            //            NSLog(@"--%ld",dispatch_semaphore_signal(semaphore));
            dispatch_semaphore_signal(semaphore);//发送一个信号，并使信号量+1

        });
        dispatch_semaphore_wait(semaphore, DISPATCH_TIME_FOREVER);//等待(看效果像是监听)信号量  如果信号量为0 则堵塞此线程，当发现信号量不为0时则继续运行并使信号量 - 1   第二个参数为等待时间     此时semaphore信号量的值如果 = 0：那么就阻塞该函数所处的线程,阻塞时长为timeout指定的时间，如果阻塞时间内semaphore的值被dispatch_semaphore_signal函数加1了，该函数所处线程获得了信号量被唤醒。然后对semaphore计数进行减1并返回，继续向下执行。 如果阻塞时间内没有获取到信号量唤醒线程或者信号量的值一直为0，那么就要等到指定的阻塞时间后，该函数所处线程才继续向下执行。
        //        dispatch_semaphore_wait(semaphore, dispatch_time(DISPATCH_TIME_NOW, NSEC_PER_SEC * 4));

    });
    dispatch_group_async(group, queue, ^{
```

```
    //        NSLog(@"2");

    dispatch_semaphore_t sema = dispatch_semaphore_create(0);
    dispatch_async(global_queue, ^{
        sleep(5);
        NSLog(@"2");
        dispatch_semaphore_signal(sema);
    });
    dispatch_semaphore_wait(sema, DISPATCH_TIME_FOREVER);

});

dispatch_group_async(group, queue, ^{
    //        NSLog(@"3");
    dispatch_semaphore_t sema = dispatch_semaphore_create(0);
    dispatch_async(global_queue, ^{
        NSLog(@"3");
        dispatch_semaphore_signal(sema);
    });
    dispatch_semaphore_wait(sema, DISPATCH_TIME_FOREVER);
});
dispatch_group_async(group, queue, ^{
    //        sleep(5);
    //        NSLog(@"4");


    dispatch_semaphore_t sema = dispatch_semaphore_create(0);
        dispatch_async(global_queue, ^{
            NSLog(@"4");

            dispatch_semaphore_signal(sema);
        });

        dispatch_semaphore_wait(sema, DISPATCH_TIME_FOREVER);
    });

//    异步监听所有任务完成
    dispatch_group_notify(group, queue, ^{
        NSLog(@"all");

    });



 /*
 @result
 * Returns zero on success (all blocks associated with the group completed
 * within the specified timeout) or non-zero on error (i.e. timed out).
 */
    //同步监听所有任务完成
//    dispatch_group_wait(group, dispatch_time(DISPATCH_TIME_NOW, NSEC_PER_SEC * 5));



    NSLog(@"主线程");

}



- (void)doSemaphore2{
    dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
    dispatch_semaphore_t semaphore = dispatch_semaphore_create(1);

    NSMutableArray *array = [[NSMutableArray alloc] init];


    for (NSInteger i = 0; i < 100; i++) {
        dispatch_async(queue, ^{
            /*
             此时semaphore信号量的值如果 >= 1时：对semaphore计数进行减1,然后dispatch_semaphore_wait 函数返回。该函数所处线程就继续执行下面的语句。

             此时semaphore信号量的值如果=0：那么就阻塞该函数所处的线程,阻塞时长为timeout指定的时间，如果阻塞时间内semaphore的值被dispatch_semaphore_signal函数加1了，该函数所处线程获得了信号量被唤醒。然后对semaphore计数进行减1并返回，继续向下执行。 如果阻塞时间内没有获取到信号量唤醒线程或者信号量的值一直为0，那么就要等到指定的阻塞时间后，该函数所处线程才继续向下执行。

             执行到这里semaphore的值总是1
             */
            dispatch_semaphore_wait(semaphore, DISPATCH_TIME_FOREVER);

            /* 因为dispatch_semaphore_create创建的semaphore的初始值为1，执行完上面的
             dispatch_semaphore_wait函数之后，semaphore计数值减1会变为0，所以可访问array对象的线程只有1个，因此可安全地对array进行操作。
             */

            [array addObject:[NSNumber numberWithInteger:i]];

//我感觉锁一下也可以

//             @synchronized (array) {
//                 [array addObject:[NSNumber numberWithInteger:i]];
//
//             }


            /*
             对array操作之后，通过dispatch_semaphore_signal将semaphore的计数值加1，此时semaphore的值由变成了1，所处
             */
            dispatch_semaphore_signal(semaphore);
```

```
        });
    }

}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}



@end
```

```
2016-12-29 16:45:47.156 GCDtest[13065:205411] 主线程
2016-12-29 16:45:53.223 GCDtest[13065:205450] 1
2016-12-29 16:45:58.298 GCDtest[13065:205450] 2
2016-12-29 16:45:58.299 GCDtest[13065:205450] 3
2016-12-29 16:45:58.299 GCDtest[13065:205450] 4
2016-12-29 16:45:58.299 GCDtest[13065:205448] all
```

举报

举报