iOS开发系列-线程同步 dispatch_barrier_(a)sync

参考

```
//参数1：将barrier添加到的队列
//参数2：barrier执行的block
void dispatch_barrier_async(dispatch_queue_t queue, dispatch_block_t block)
void dispatch_barrier_sync(dispatch_queue_t queue, dispatch_block_t block);
```

官方文档关于该函数的文档说明

说明：

共同点：

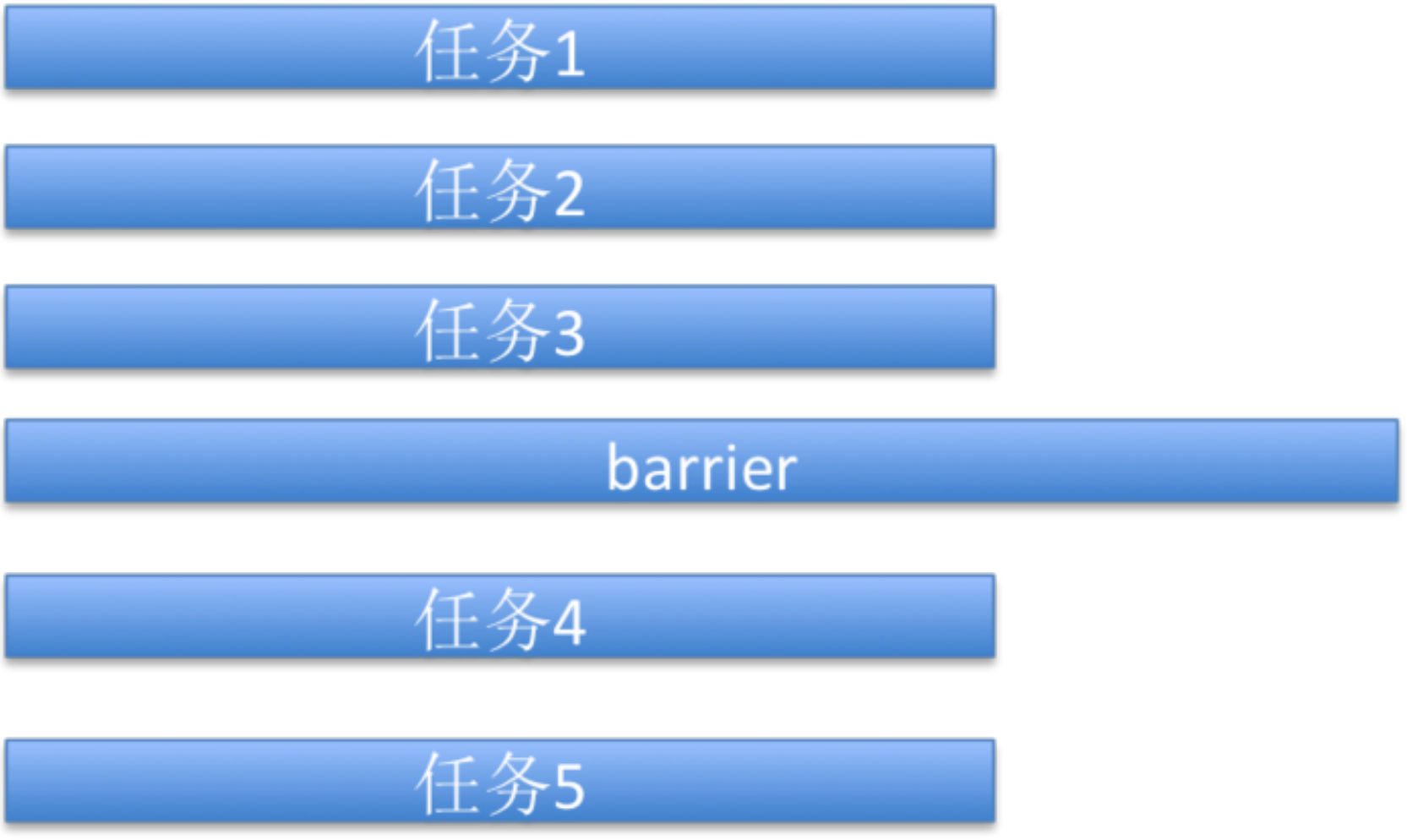
1、等待在它前面插入队列的任务先执行完

2、等待他们自己的任务执行完再执行后面的任务

不同点：

1、dispatch_barrier_sync将自己的任务插入到队列的时候，需要等待自己的任务结束之后才会继续插入被写在它后面的任务，然后执行它们

2、dispatch_barrier_async将自己的任务插入到队列之后，不会等待自己的任务结束，它会继续把后面的任务插入到队列，然后等待自己的任务结束后才执行后面任务。



barrier.png

图例说明：在添加到队列的任务1、任务2、任务3执行完毕后，然后才执行barrier，barrier执行完毕后才执行任务4、任务5

代码如下：dispatch_barrier_async

```
- (void)concurrentQueueBarrier{
    //1 创建并发队列
    dispatch_queue_t concurrentQueue = dispatch_queue_create("concurrentQue

    //2 向队列中添加任务
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务1,%@",[NSThread currentThread]);
    });
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务2,%@",[NSThread currentThread]);
    });
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务3,%@",[NSThread currentThread]);
    });
    dispatch_barrier_async(concurrentQueue, ^{
        [NSThread sleepForTimeInterval:1.0];
        NSLog(@"我是barrier");
    });
    NSLog(@"aa");
```

```
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务4,%@",[NSThread currentThread]);
    });
    NSLog(@"bb");
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务5,%@",[NSThread currentThread]);
    });

}
```

执行结果:

任务3,<NSThread: 0x600000078640>{number = 5, name = (null)}

aa

任务1,<NSThread: 0x600000078600>{number = 3, name = (null)}

任务2,<NSThread: 0x60800007cd00>{number = 4, name = (null)}

bb

我是barrier

任务4,<NSThread: 0x60800007cd00>{number = 4, name = (null)}

任务5,<NSThread: 0x600000078600>{number = 3, name = (null)}

代码如下：dispatch_barrier_sync

```
- (void)concurrentQueueBarrier{
    //1 创建并发队列
    dispatch_queue_t concurrentQueue = dispatch_queue_create("concurrentQue

    //2 向队列中添加任务
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务1,%@",[NSThread currentThread]);
    });
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务2,%@",[NSThread currentThread]);
    });
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务3,%@",[NSThread currentThread]);
    });
    dispatch_barrier_sync(concurrentQueue, ^{
        [NSThread sleepForTimeInterval:1.0];
        NSLog(@"我是barrier");
    });
    NSLog(@"aa");

    dispatch_async(concurrentQueue, ^{
```

```
        NSLog(@"任务4,%@",[NSThread currentThread]);
    });
    NSLog(@"bb");
    dispatch_async(concurrentQueue, ^{
        NSLog(@"任务5,%@",[NSThread currentThread]);
    });

}
```

执行结果：

任务3,<NSThread: 0x600000078640>{number = 5, name = (null)}

任务1,<NSThread: 0x600000078600>{number = 3, name = (null)}

任务2,<NSThread: 0x60800007cd00>{number = 4, name = (null)}

我是barrier

aa

bb

任务4,<NSThread: 0x60800007cd00>{number = 4, name = (null)}

任务5,<NSThread: 0x600000078600>{number = 3, name = (null)}

好了，说到这应该差不多能想通了，我们开始总结

dispatch_barrier_sync和dispatch_barrier_async的共同点：

1、都会等待在它前面插入队列的任务（1、2、3）先执行完

2、都会等待他们自己的任务（0）执行完再执行后面的任务（4、5、6）

dispatch_barrier_sync和dispatch_barrier_async的不共同点：

在将任务插入到queue的时候，dispatch_barrier_sync需要等待自己的任务（0）结束之后才会继续程序，然后插入被写在它后面的任务（4、5、6），然后执行后面的任务

而dispatch_barrier_async将自己的任务（0）插入到queue之后，不会等待自己的任务结束，它会继续把后面的任务（4、5、6）插入到queue

所以，dispatch_barrier_async的不等待（异步）特性体现在将任务插入队列的过程，它的等待特性体现在任务真正执行的过程

翻译：

调用这个函数总是在barrier block被提交之后立即返回，不会等到block被执行。当barrier block到并发队列的最前端，他不会立即执行。相反，队列会等到所有当前正在执行的blocks结束执行。到这时，barrier才开始自己执行。所有在barrier block之后提交的blocks会等到barrier block结束之后才执行。

这里指定的并发队列应该是自己通过dispatch_queue_create函数创建的。如果你传的是一个串行队列或者全局并发队列，这个函数等同于dispatch_async函数。