GCD使用经验与技巧浅谈

2015-05-05 09:01 编辑：lansekuangtu 分类：iOS开发 来源：土土哥的技术Blog 5 16331

iOS开发 GCD

前言

GCD(Grand Central Dispatch)可以说是Mac、iOS开发中的一大“利器”，本文就总结一些有关使用GCD的经验与技巧。

dispatch_once_t必须是全局或static变量

这一条算是“老生常谈”了，但我认为还是有必要强调一次，毕竟非全局或非static的dispatch_once_t变量在使用时会导致非常不好排查的bug，正确的如下：

```
//静态变量，保证只有一份实例，才能确保只执行一次
static dispatch_once_t onceToken;
dispatch_once(&onceToken, ^{
    //单例代码
});
```

其实就是保证**dispatch_once_t**只有一份实例。

dispatch_queue_create的第二个参数

dispatch_queue_create，创建队列用的，它的参数只有两个，原型如下：

```
dispatch_queue_t dispatch_queue_create ( const char *label, dispatch_queue_attr_t attr )
```

在网上的大部分教程里（甚至Apple自己的文档里），都是这么创建串行队列的：

```
dispatch_queue_t queue = dispatch_queue_create("com.example.MyQueue", NULL);
```

看，第二个参数传的是“**NULL**”。 但是dispatch_queue_attr_t类型是有已经定义好的常量的，所以我认为，为了更加的清晰、严谨，最好如下创建队列：

```
//串行队列
dispatch_queue_t queue = dispatch_queue_create("com.example.MyQueue", DISPATCH_QUEUE_SERIAL
//并行队列
dispatch_queue_t queue = dispatch_queue_create("com.example.MyQueue", DISPATCH_QUEUE_CONCUR
```

常量就是为了使代码更加“易懂”，更加清晰，既然有，为啥不用呢~

dispatch_after是延迟提交，不是延迟运行

热门资讯


iOS项目分析及优化
点击量 8243


iOS-图文表并茂，手把手教你GCD
点击量 6452


做一个 App 前需要考虑的几件事
点击量 6352


苹果MBP接口太超前 理想主义总要付点代价
点击量 5864


一触即发：MacBook Pro全新阵容亮相
点击量 5542


一个iOS模块化开发解决方案
点击量 5110


如何优化 App 的启动时间
点击量 4540


iOS开发系列--App扩展开发
点击量 4306


三款全新MacBook型号曝光 MacBook Air真
点击量 3728


迭代器模式(Java与IOS)
点击量 3693

综合评论

呵呵，你能考虑到的 人家没有考虑过? 你想不到的 人家 也 想不到? 人家 发我贱了 评论了 微信小程序的想象力与不可想象域...

安卓风，能不能搞个iOS系统的分析分析
Viterbi 评论了 UI设计中下拉刷新有什么讲究? ...

反正磁吸充电口没了真是不能忍
wengxianxun 评论了 苹果大幅下调USB-C配件价格是被舆论所逼? ...

先看看官方文档的说明：

```
Enqueue a block for execution at the specified time.
```

Enqueue，就是入队，指的就是将一个Block在特定的延时以后，加入到指定的队列中，不是在特定的时间后立即运行！。

看看如下代码示例：

```
建串行队列
atch_queue_t queue = dispatch_queue_create("me.tutuge.test.gcd", DISPATCH_QUEUE_CONCURRENT);
即打印一条信息
g(@"Begin add block...");
交一个block
atch_async(queue, ^{
//Sleep 10秒
[NSThread sleepForTimeInterval:10];
NSLog(@"First block done...");

秒以后提交block
atch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(5 * NSEC_PER_SEC)), queue, ^{
NSLog(@"After...");
```

结果如下：

```
2015-03-31 20:57:27.122 GCDTest[45633:1812016] Begin add block...
2015-03-31 20:57:37.127 GCDTest[45633:1812041] First block done...
2015-03-31 20:57:37.127 GCDTest[45633:1812041] After...
```

从结果也验证了，dispatch_after只是延时提交block，并不是延时后立即执行。所以想用dispatch_after精确控制运行状态的朋友可要注意了~

正确创建dispatch_time_t

用dispatch_after的时候就会用到dispatch_time_t变量，但是如何创建合适的时间呢？答案就是用dispatch_time函数，其原型如下：

```
dispatch_time_t dispatch_time ( dispatch_time_t when, int64_t delta );
```

第一个参数一般是**DISPATCH_TIME_NOW**，表示从现在开始。

那么第二个参数就是真正的延时的具体时间。

这里要特别注意的是，**delta**参数是“纳秒！”，就是说，延时1秒的话，delta应该是“1000000000”=。=，太长了，所以理所当然系统提供了常量，如下：

```
#define NSEC_PER_SEC 1000000000ull
#define USEC_PER_SEC 1000000ull
#define NSEC_PER_USEC 1000ull
```

关键词解释：

- NSEC：纳秒。
- USEC：微妙。
- SEC：秒
- PER：每

所以：

1. NSEC_PER_SEC，每秒有多少纳秒。
2. USEC_PER_SEC，每秒有多少毫秒。（注意是指在纳秒的基础上）

3. NSEC_PER_USEC，每毫秒有多少纳秒。

所以，延时**1**秒可以写成如下几种：

dispatch_time(DISPATCH_TIME_NOW, 1 * NSEC_PER_SEC);

dispatch_time(DISPATCH_TIME_NOW, 1000 * USEC_PER_SEC);

dispatch_time(DISPATCH_TIME_NOW, USEC_PER_SEC * NSEC_PER_USEC);

最后一个“**USEC_PER_SEC * NSEC_PER_USEC**”，翻译过来就是“每秒的毫秒数乘以每毫秒的纳秒数”，也就是“每秒的纳秒数”，所以，延时500毫秒之类的，也就不难了吧~

dispatch_suspend != 立即停止队列的运行

dispatch_suspend，**dispatch_resume**提供了“挂起、恢复”队列的功能，简单来说，就是可以暂停、恢复队列上的任务。但是这里的“挂起”，并不能保证可以立即停止队列上正在运行的block，看如下例子：

```
dispatch_queue_t queue = dispatch_queue_create("me.tutuge.test.gcd", DISPATCH_QUEUE_SEF
//提交第一个block，延时5秒打印。
dispatch_async(queue, ^{
    [NSThread sleepForTimeInterval:5];
    NSLog(@"After 5 seconds...");
});
//提交第二个block，也是延时5秒打印
dispatch_async(queue, ^{
    [NSThread sleepForTimeInterval:5];
    NSLog(@"After 5 seconds again...");
});
//延时一秒
NSLog(@"sleep 1 second...");
[NSThread sleepForTimeInterval:1];
//挂起队列
NSLog(@"suspend...");
dispatch_suspend(queue);
//延时10秒
NSLog(@"sleep 10 second...");
[NSThread sleepForTimeInterval:10];
//恢复队列
NSLog(@"resume...");
dispatch_resume(queue);
```

运行结果如下：

```
2015-04-01 00:32:09.903 GCDTest[47201:1883834] sleep 1 second...
2015-04-01 00:32:10.910 GCDTest[47201:1883834] suspend...
2015-04-01 00:32:10.910 GCDTest[47201:1883834] sleep 10 second...
2015-04-01 00:32:14.908 GCDTest[47201:1883856] After 5 seconds...
2015-04-01 00:32:20.911 GCDTest[47201:1883834] resume...
2015-04-01 00:32:25.912 GCDTest[47201:1883856] After 5 seconds again...
```

可知，在dispatch_suspend挂起队列后，第一个block还是在运行，并且正常输出。

结合文档，我们可以得知，dispatch_suspend并不会立即暂停正在运行的block，而是在当前block执行完成后，暂停后续的block执行。

所以下次想暂停正在队列上运行的block时，还是不要用dispatch_suspend了吧~

“同步”的dispatch_apply

dispatch_apply的作用是在一个队列（串行或并行）上“运行”多次block，其实就是简化了用循环去向队列依次添加block任务。但是我个人觉得这个函数就是个“坑”，先看看如下代码运行结果：

```
//创建异步串行队列
dispatch_queue_t queue = dispatch_queue_create("me.tutuge.test.gcd", DISPATCH_QUEUE_SERI
//运行block3次
```

```
dispatch_apply(3, queue, ^(size_t i) {
    NSLog(@"apply loop: %zu", i);
});
//打印信息
NSLog(@"After apply");
```

运行的结果是：

```
2015-04-01 00:55:40.854 GCDTest[47402:1893289] apply loop: 0
2015-04-01 00:55:40.856 GCDTest[47402:1893289] apply loop: 1
2015-04-01 00:55:40.856 GCDTest[47402:1893289] apply loop: 2
2015-04-01 00:55:40.856 GCDTest[47402:1893289] After apply
```

看，明明是提交到异步的队列去运行，但是“After apply”居然在apply后打印，也就是说，dispatch_apply将外面的线程（main线程）“阻塞”了！

查看官方文档，dispatch_apply确实会“等待”其所有的循环运行完毕才往下执行=。=，看来要小心使用了。

避免死锁！

dispatch_sync导致的死锁

涉及到多线程的时候，不可避免的就会有“死锁”这个问题，在使用GCD时，往往一不小心，就可能造成死锁，看看下面的“死锁”例子：

```
//在main线程使用“同步”方法提交Block，必定会死锁。
dispatch_sync(dispatch_get_main_queue(), ^{
    NSLog(@"I am block...");
});
```

你可能会说，这么低级的错误，我怎么会犯，那么，看看下面的：

```
- (void)updateUI1 {
    dispatch_sync(dispatch_get_main_queue(), ^{
        NSLog(@"Update ui 1");

        //死锁!
        [self updateUI2];
    });
}
- (void)updateUI2 {
    dispatch_sync(dispatch_get_main_queue(), ^{
        NSLog(@"Update ui 2");
    });
}
```

在你不注意的时候，嵌套调用可能就会造成死锁！所以为了“世界和平”=。=，我们还是少用dispatch_sync吧。

dispatch_apply导致的死锁！

啥，dispatch_apply导致的死锁？。。。是的，前一节讲到，dispatch_apply会等循环执行完成，这不就差不多是阻塞了吗。看如下例子：

```
dispatch_queue_t queue = dispatch_queue_create("me.tutuge.test.gcd", DISPATCH_QUEUE_SEF

dispatch_apply(3, queue, ^(size_t i) {
NSLog(@"apply loop: %zu", i);

    //再来一个dispatch_apply! 死锁!
dispatch_apply(3, queue, ^(size_t j) {
NSLog(@"apply loop inside %zu", j);
});
});
```

这端代码只会输出“apply loop: 1”。。。就没有然后了=。=

所以，一定要避免dispatch_apply的嵌套调用。

灵活使用dispatch_group

很多时候我们需要等待一系列任务（block）执行完成，然后再做一些收尾的工作。如果是有序的任务，可以分步骤完成的，直接使用串行队列就行。但是如果是一系列并行执行的任务呢？这个时候，就需要dispatch_group帮忙了~总的来说，dispatch_group的使用分如下几步：

1. 创建dispatch_group_t
2. 添加任务（block）
3. 添加结束任务（如清理操作、通知UI等）

下面着重讲讲在后面两步。

添加任务

添加任务可以分为以下两种情况：

自己创建队列：使用**dispatch_group_async**。

无法直接使用队列变量（如使用AFNetworking添加异步任务）：使用**dispatch_group_enter**，**dispatch_group_leave**。

自己创建队列时，当然就用dispatch_group_async函数，简单有效，简单例子如下：

```
//省去创建group、queue代码。。。
dispatch_group_async(group, queue, ^{
    //Do you work...
});
```

当你无法直接使用队列变量时，就无法使用dispatch_group_async了，下面以使用AFNetworking时的情况：

```
AFHTTPRequestOperationManager *manager = [AFHTTPRequestOperationManager manager];
//Enter group
dispatch_group_enter(group);
[manager GET:@"http://www.baidu.com" parameters:nil success:^(AFHTTPRequestOperation *c
    //Deal with result...
    //Leave group
    dispatch_group_leave(group);
}    failure:^(AFHTTPRequestOperation *operation, NSError *error) {
    //Deal with error...
    //Leave group
    dispatch_group_leave(group);
}];
//More request...
```

使用**dispatch_group_enter**，**dispatch_group_leave**就可以方便的将一系列网络请求“打包”起来~

添加结束任务

添加结束任务也可以分为两种情况，如下：

1. 在当前线程阻塞的同步等待：dispatch_group_wait。
2. 添加一个异步执行的任务作为结束任务：dispatch_group_notify

这两个比较简单，就不再贴代码了=。=

使用dispatch_barrier_async,dispatch_barrier_sync的注意事项

dispatch_barrier_async的作用就是向某个队列插入一个block，当目前正在执行的block运行完成后，阻塞这个block后面添加的block，只运行这个block直到完成，然后再继续后续的任务，有点“唯我独尊”的感觉=。=

值得注意的是：

dispatchbarrier\(a)sync只在自己创建的并发队列上有效，在全局(Global)并发队列、串行队列上，效果跟**dispatch_(a)sync**效果一样。

既然在串行队列上跟dispatch_(a)sync效果一样，那就要小心别死锁！

dispatch_set_context与dispatch_set_finalizer_f的配合使用

dispatch_set_context可以为队列添加上下文数据，但是因为GCD是C语言接口形式的，所以其context参数类型是“void *”。也就是说，我们创建context时有如下几种选择：

用C语言的malloc创建context数据。

用C++的new创建类对象。

用Objective-C的对象，但是要用__bridge等关键字转为Core Foundation对象。

以上所有创建context的方法都有一个必须的要求，就是都要释放内存！，无论是用free、delete还是CF的CFRelease，我们都要确保在队列不用的时候，释放context的内存，否则就会造成内存泄露。

所以，使用dispatch_set_context的时候，最好结合dispatch_set_finalizer_f使用，为队列设置“析构函数”，在这个函数里面释放内存，大致如下：

```
void cleanStaff(void *context) {
    //释放context的内存!
    //CFRelease(context);
    //free(context);
    //delete context;
}
...
//在队列创建后，设置其“析构函数”
dispatch_set_finalizer_f(queue, cleanStaff);
```

详细用法，请看我之前写的Blog为GCD队列绑定NSObject类型上下文数据-利用__bridge_retained(transfer)转移内存管理权

总结

其实本文更像是总结了GCD中的“坑”=。=

至于经验，总结一条，就是使用任何技术，都要研究透彻，否则后患无穷啊~

参考

- Grand Central Dispatch (GCD) Reference
- Concurrency Programming Guide
- Using Dispatch Groups to Wait for Multiple Web Services

上一篇：源码推荐(5.07):JsWebView实现objectc与javascript交互，高仿支付宝手势解锁（增强版）

下一篇：几个iOS工程通用模块介绍

相关资讯

- iOS开发网络——数据缓存
- Objc 对象的今生今世
- iOS开发中的这些权限，你搞懂了吗?
- 巧谈GCD
- iOS-图文表并茂，手把手教你GCD
- iOS10 从Safari跳转到描述文件与设备管理
- 深入研究Block捕获外部变量和__block实现原理
- iOS开发系列--App扩展开发
- iOS开发之玩转蓝牙CoreBluetooth
- iOS开发之Xcode常用调试技巧总结

我来说两句

你怎么看？快来评论一下吧!

发表评论

所有评论（5）

G_Hand 2016-08-04 16:04:53

1*NSEC_PER_SEC 是1秒吗？无法理解啊，大神能用数字表达一下吗，让我搞明白。。

0 0 回复

LarryHwang 2016-07-29 10:51:06

分享的很深刻，但写的很垃圾 谢谢

3 1 回复

麦的守护 2016-04-25 03:27:41

讲的很详细,也很全面,赞一个!!

0 0 回复

JohnHou 2016-04-21 03:20:04

果然 大神

0 0 回复

hms111111 2016-03-15 08:15:25

我被搞晕了,一会注释是窜行,一会是并行,

1 0 回复