作者 _叫我小贱 (/users/e78a53f948a3) 2016.06.26 11:26*
写了16822字，被5人关注，获得了6个喜欢
(/users/e78a53f948a3)

iOS多线程－GCD的使用

GCD是通过C语言编写的，为多核并行运算提出了解决方案。在GCD中，GCD会自动利用多核处理器并且自动管理线程的生命周期（线程的创建、调度、销毁）。

- GCD用有任务和队列两个基本概念
    - 任务：就是你想在线程中要做的事情。
    - 队列：GCD会将遵循FIFO（操作系统中的概念）的原则，将队列中的任务取出放到对应线程中去执行。

- GCD简单的使用格式

```
//同步方式执行任务
dispatch_sync(dispatch_queue_t queue, ^(void)block);
//异步方式执行任务
dispatch_async(dispatch_queue_t queue,^(void)block);
```

    - queue：任务添加到的队列 ,block：执行任务的代码。
    - 同步不能开启一个新线程，只能在当前线程中执行任务。
    - 异步能开启一个新的线程，并在新线程中执行任务。(异步函数用在主队列上不会开新的线程)
    - queue的类型
        - 并发队列：多个任务并发执行。并发功能只在dispatch_async中有效。
        - 串行队列：任务一个接着一个的执行。

- 队列的获得：

1.创建一个队列:

```
//创建队列queue
dispatch_queue_t queue = dispatch_queue_create(const char *label,   dispatch_queue_attr_t
//释放队列（非ARC）
dispatch_release(queue);
```

- label:队列的名称，一般用功能命名。
- attr:队列的类型:
  - DISPATCH_QUEUE_SERIAL 串行队列。
  - DISPATCH_QUEUE_CONCURRENT 并发队列。

2.使用GCD提供的全局并发队列，整个应用都可以使用。（并发队列）

```
dispatch_queue_t queue = dispatch_get_global_queue(long identifier, unsigned long flags);
```

- identifier: 优先级。(由高到低)
  - DISPATCH_QUEUE_PRIORITY_HIGH 2
  - DISPATCH_QUEUE_PRIORITY_DEFAULT 0 （一般使用这个）
  - DISPATCH_QUEUE_PRIORITY_LOW (-2)
  - DISPATCH_QUEUE_PRIORITY_BACKGROUND INT16_MIN
- flags:官方文档上写的是 `Flags that are reserved for future use. Always specify 0 for this parameter.` 所以我们写0就好了。

3.主队列（串行队列）

- 主队列的任务会放到主线程中去执行。

```
dispatch_queue_t queue = dispatch_get_main_queue();
//获得主队列
```

4.总结:

- 并发队列获得的两种方式
  - dispatch_queue_create，队列参数：DISPATCH_QUEUE_CONCURRENT
  - dispatch_get_global_queue
- 串行队列获得的两种方式
  - dispatch_queue_create, 队列参数：DISPATCH_QUEUE_SERIAL
  - dispatch_get_main_queue

5.同步函数和异步函数的区别

```
-(void)sync
{
    dispatch_queue_t queue =     dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFA

    dispatch_sync(queue, ^{
        NSLog(@"1-%@",[NSThread currentThread]);
    });

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

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

    NSLog(@"END");
}
-(void)async
{
    dispatch_queue_t queue =     dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFA

    dispatch_async(queue, ^{
         NSLog(@"1-%@",[NSThread currentThread]);
     });

     dispatch_async(queue, ^{
         NSLog(@"1-%@",[NSThread currentThread]);
     });

     dispatch_async(queue, ^{
         NSLog(@"1-%@",[NSThread currentThread]);
     });

     NSLog(@"END");
 }
```

由此可知同步函数（sync）要执行完线程，才能执行完当前任务。异步函数（async）执行完当前任务，再执行任务里的线程。

6.不同的队列和同步函数／异步函数所产生的效果

队列的类型	并发队列	手动创建的串行队列	主队列

同步(sync)	不开启新线程，串行执行任务	不开启新线程，串行执行任务	不开启新线程，串行执行任务
异步(async)	开启新线程，并发执行任务	开启新线程，串行执行任务	不开启新线程，串行执行任务

- 使用sync(同步函数)往当前串行队列添加任务，会使当前串行队列侵入死循环。

- 从子线程回到主线程

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0),     ^{
        //子线程中执行的操作

        dispatch_async(dispatch_get_main_queue(), ^{
            //回到主线程的操作
        });
    });
```

- dispatch_barrier的使用

```
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，它后面的任务要等到它执行完才会执行。Apple官方文档里这样描     述：Th
```

- 延时函数

```
- (void)performSelector:(SEL)aSelector withObject:(nullable id)anArgument afterDelay:(NSTime
//delay为延时时间,延时1s就写1.0
//通过runloop实现

dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(delayInSeconds * NSEC_PER_SEC)), d
        code to be executed after a specified delay
    });
// delayInSeconds为延时时间,延时1s就写1.0
//dispatch_get_main_queue() 代表在主队列中执行任务，如果想在子线程中执行任务，把此队列改为并发队列和串行队

[NSTimer scheduledTimerWithTimeInterval:(NSTimeInterval) target:(nonnull id) selector:(nonnu
//NSTimeInterval为延时时间,延时1s就写1.0
//nonnull id 为self
//nullable id 为nil
//repeats:代表是否重复，不重复填NO
//通过runloop实现
```

- 一次性代码
  - dispatch_once函数能使某段代码，在程序运行过程中只执行一次，不同于懒加载（懒加载每次生成新对象都要使用），它可以用于加载程序里只能加载一次的资源。

```
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        code to be executed once
    });
    //onceToken是一个标记，标记代码有没有执行过，不要修改
```

- 快速迭代(遍历)

```
    dispatch_apply(size_t iterations, dispatch_queue_t queue, ^(size_t index) {
        code
    });
//iterations 迭代次数。迭代2次就写2。
//queue 队列。
//index 索引。
```

- 队列组

- 如过一个需求是先完成一个或者多个异步操作，完成了它们之后才能执行下一个异步操作，此时我们就需要队列组。

1.创建一个组

```
dispatch_group_t group = dispatch_group_create();
```

2.添加到组

```
dispatch_group_async(dispatch_group_t  _Nonnull group, dispatch_queue_t  _Nonnull queu
```

- group是你创建的组，queue为你创建的队列，block为执行任务的代码。
- 多个任务就要多次调用此函数

3.组里的任务都执行完之后dispatch_group_notify函数

```
dispatch_group_notify(dispatch_group_t  _Nonnull group, dispatch_queue_t  _Nonnull que
```

- group为执行完任务的组，queue在哪个任务执行的队列，block为执行任务的代码。
- 如果之前的任务在queue队列中执行，我最后想回到主线程，那么在queue的参数为dispatch_get_main_queue()即可。

0条评论 （ 按时间正序 · 按时间倒序 · 按喜欢排序 ）

添加新评论

关闭评论 (/notes/f7efbd822d7e/toggle_comment)

写下你的评论...

发 表 ☺ ⌘+Return 发表