线程

一、NSThread线程---OC

创建线程的方法：

1、用类方法创建线程

（1）[NSThread detachNewThreadSelector:(SEL)selector toTarget:(id)target withObject:(nullable id)argument;]

e.g.

[NSThread detachNewThreadSelector:@selector(thread1) toTarget: test_obj withObject:nil];

用类方法创建一个线程，其中Target线程方法从属的对象，Selector是对象里的方法；Target和Selector的关系就是新开线程执行的是Target对象里的selector方法，argument是方法的参数，必须是对象。

（2）[NSThread detachNewThreadWithBlock:(^)(void))block];

e.g.

```
[NSThread detachNewThreadWithBlock:^{
      [NSThread sleepForTimeInterval:0.5];
      NSLog(@"new thread");
  }];
```

block里面是线程执行的代码段，不用另外写函数。

类方法建立线程要记得在死循环里这只线程结束条件，线程退出**[NSThread exit]**。

2、用对象方法创建线程

NSThread thread1 = [[NSThread alloc] initWithTarget:test_obj selector:@selector(thread3) object:nil];

3、线程安全

（1）两个线程不加锁的情况下同时修改一个资源，结果不可预料，导致两个线程都修改不成功，但是可以同时读。锁里最好不要用循环，不要在锁里执行太久，会占用太多系统资源。

（2）不要在线程里死循环，一定要有退出循环的条件。

（3）不要阻塞主线程。

（4）循环一定要释放系统资源（NSRunLoop消息机制的处理模式）。

（5）让线程正常结束。

二、系统调度（GCD）

1、同步调度：

dispatch_sync(dispatch_queue_t _Nonnull queue, ^(void)block)

e.g.

```
dispatch_sync(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
    int i = 9;
    while (1) {
        if (i>0) {
            i--;
            NSLog(@"this is a thread1！ ");
        }else
        {
            break;
```

```
        }
    }
});
```

2、异步调度
dispatch_async(dispatch_queue_t _Nonnull queue, ^(void)block)
e.g.

```
// 获得默认任务队列
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,
0);
//添加一个任务到任务队列
   dispatch_async(queue, ^{
//线程执行的内容
      int i = 9;
       while (1) {
          if (i>0) {
             i--;
             NSLog(@"this is a thread1！ ");
          }else
          {
             break;
          }
    }
});
```

dispatch_async/dispatch_sync的第一个参数是任务队列dispatch_queue_t，每个程序系统自动提供三个Concurrent Queues(并行队列):

```
dispatch_queue_t  aQueue =
dispatch_get_globlal_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,0);

dispatch_queue_t  aHQueue =
dispatch_get_globlal_queue(DISPATCH_QUEUE_PRIORITY_HIGHT,0);

dispatch_queue_t  aLQueue =
dispatch_get_globlal_queue(DISPATCH_QUEUE_PRIORITY_LOW,0);
```

三个不同优先级别的concurrent queues.

dispatch_queue_t mainQueue = dispatch_get_main_queue(); 程序启动自动生成。

自己创建队列：

**并行队列：

```
dispatch_queue_t my_queue =
dispatch_queue_create(“myQueue”,DISPATCH_QUEUE_CONCRRENT);
```

**串行队列

```
dispatch_queue_t my_queue = dispatch_queue_create(“myQueue”,DISPATCH_QUEUE_SERIAL);
```

3、同步异步混合使用

（1）可以使用group管理多个并行队列完成

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT;
dispatch_group_t group = dispatch_group_create();

//add a task to the group
dispatch_group_async(group, queue, ^{printf(“task 1 \n”); });
//add a task to the group
```

```
dispatch_group_async(group, queue, ^{printf("task 2 \n"); });
printf("wait task1,2\n");
dispatch_group_wait(group,DISPATCTH_TIME_OFREVER);
print("task a,2 finished\n");
dispatch_release(group);

//create new group
group = dispatch_group_create();
//add tasks to group
dispatch_group_async(group, queue, ^{printf("task 3 \n"); });
printf("wait task1,2\n");
dispatch_group_wait(group,DISPATCTH_TIME_OFREVER);
print("task a,2 finished\n");
dispatch_release(group);
```

这样完成task1 和task 2的同步，而task1，2和task 3异步。

（2）可以在queue中定义一个结束函数

4、信号量（semaphore）控制并发

信号量操作的三个函数：

dispatch_semaphore_create　创建信号量。创建n个信号量，代表可同时运行的最多线程数

dispatch_semaphore_signal　发送一个新号。信号量+1

dispatch_semaphore_wait　　等待信号。信号量小于0会一直等待，否则正常执行，信号量-1

可以使用group完成

```
dispatch_group_t group = dispatch_group_create();
    dispatch_semaphore_t semaphore = dispatch_semaphore_create(10);
    dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
    for (int i = 0; i < 100; i++)
    {
        dispatch_semaphore_wait(semaphore, DISPATCH_TIME_FOREVER);
        dispatch_group_async(group, queue, ^{
            NSLog(@"%i",i);
            sleep(2);
            dispatch_semaphore_signal(semaphore);
        });
    }
    dispatch_group_wait(group, DISPATCH_TIME_FOREVER);
    dispatch_release(group);
    dispatch_release(semaphore);
```