多线程

1. 简介：
2. 一个应用对应一个进程，负责申请一块内存，每个进程启动后有一个主线程会开启。线程是程序的执行流。同一个进程中多个线程共用进程的内存空间。
3. 遇到耗时的操作时，新建线程去执行可保证主线程的流畅
4. 一开始多线程用于单核处理并发任务，由CPU整体调度；随着多核处理器的出现，新建线程时，具体在哪个CPU上执行任务，由系统调度
5. 主线程是其他线程的父线程，负责所有界面显示操作和相应用户点击事件
6. 每创建一个线程，都会消耗一定的内存和CPU时间(系统资源)
7. 当多线程对同一资源进行抢夺时需注意线程安全问题
8. 线程之间要控制执行顺序相对比较复杂，以及共享资源的争夺问题
9. 多线程是为了并发同步完成多项任务，而不是为了提高运行效率，是为了通过提高资源使用效率提高系统的整体性能，不会提高单个算法本身的执行时间
10. 线程使用不是无节制的：主线程堆栈大小1M，其他线程512K，且不能修改其值
11. 的NSThread:轻量级，使用简单，需自己管理线程生命周期、线程同步、加锁、睡眠等。线程同步对数据的加锁会有一定的系统开销

NSOperation:不需要关心线程管理，数据同步问题，可把精力放在自己要执行的操作上，NSOperation是面向对象的

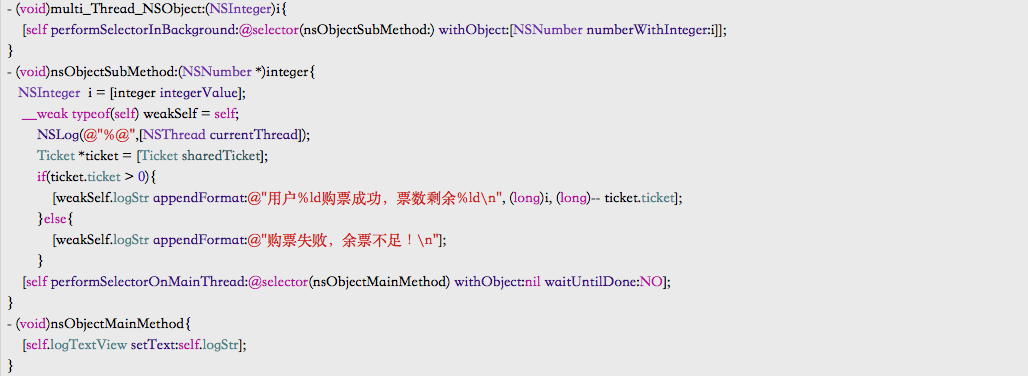
GCD:苹果开发的一个多核编程的解决方案。替代NSOperation、NSThread的高效和强大的技术，基于C语言

1. 多线程的使用
2. NSObject封装的基于NSThread的简单多线程方法

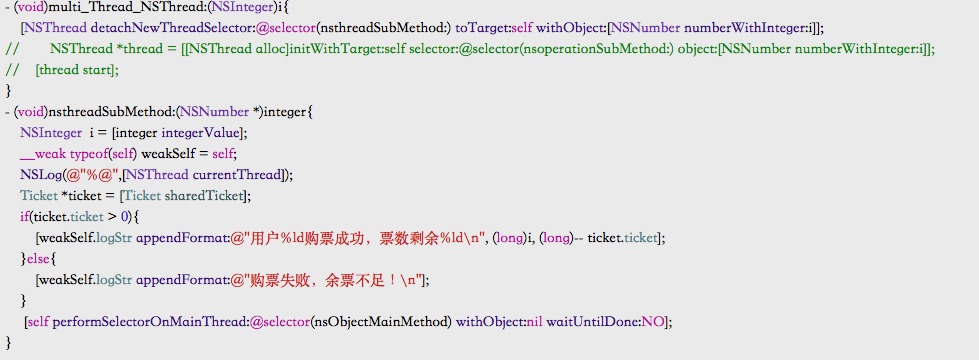
waitUntilDown:当前线程是否需要被阻塞，直到主线程将我们指定的代码执行完

注意：当前线程为主线程时，waitUntilDown参数无效，UI一般是回到主线程进行修改的

内存管理：线程任务需要包裹在@autoreleasepool中，否者容易出现内存泄漏



1. NSThread多线程:



缺点：控制线程生命周期难，控制并发线程数难，控制线程先后顺序难

1. NSOperation&NSOperationQueue



队列设置并发进程数

[self.queue setMaxConCurrentOperationCount:2]

线程执行顺序控制

[op2 addDependcy:op1];

注：不要建立循环依赖，否者系统不崩溃也不干活，依赖可以跨队列

1. GCD技术
2. 核心理念：针对多核处理器定制的，FIFO队列，dispatch queue，可保证先来的任务先执行
3. 几种队列

全局队列：所有添加到全局队列到任务都是并发执行的

串行队列：所有添加到串行队列的任务都是顺序执行的

主队列：所有添加到主队列中的任务都是在主线程中执行的

1. 派发dispatch

//异步async执行，并发执行

dspatch\_queue\_t queue = dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT,0);//0保留属性

dispatch\_async(queue, ^{

//下载图片后

dispatch\_async(dispatch\_get\_main\_queue(),^{

[\_ setImage:image];

});  
});

1. 串行队列：需创建，不能get

dispatch\_queue\_t = dispatch\_queue\_create(“”, DISPATCH\_QUEUE\_SERIAL);

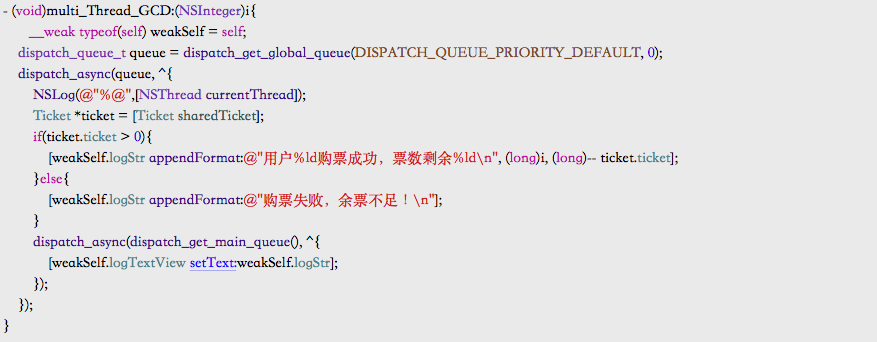
dispatch\_async(queue, ^{

//任务1  
})

dispatch\_async(queue, ^{

//任务2  
})

//结果1->2顺序执行



1. 注意点
2. 串行队列异步任务，会使用同一个子线程依此执行
3. 并行队列的异步任务会使用多个子线程无序执行
4. 全局队列类似并行队列，但无法命名，调试，无法确认任务所在队列
5. 串行队列中的任务只会顺序执行
6. 并行队列：队列中的任务通常会并发执行
7. dispatch\_async：异步操作，会并发执行，无法确定任务顺序
8. dspatch\_sync：同步操作，会依次执行，能决定任务的执行顺序
9. 串行队列同步操作，操作不会新建线程，操作是顺序执行的
10. 串行队列异步操作，新建一个子线程顺序执行，是最安全的选择
11. 并行队列同步操作，操作不会新建线程，操作顺序执行
12. 并行队列异步操作，新建线程，操作无序执行
13. 主队列异步：操作都在主线程上顺序执行，不存在异步
14. 主队列同步：除非主线程被干掉，否者永远不会结束，主队列中添加的同步操作永远也不会执行，会死锁
15. 主队列异步虽然不开新线程，但会把异步任务降低优先级，空闲时执行
16. 同步任务执行完返回，异步任务直接返回。同步会阻塞当前线程，并把任务丢到相应的队列，并等待完成。
17. GCD死锁案例
18. 当同步遇到了串行

NSLog(@“1”);

dipatch\_sync(dispatch\_get\_main\_queue(),^{

NSLog(@“2”);

})

NSLog(@“3”)；

结果：输出1

1. 当同步遇到了并行

NSLog(@“1”);

dipatch\_sync(dispatch\_get\_global\_queue(...),^{

NSLog(@“2”);

})

NSLog(@“3”)；

结果：123顺序输出

1. 同步与异步遇到了串行

dispatch\_queue\_t queue = dispatch\_queue\_create(“”, ...SERIAL);

NSLog(@“1”)；

dispatch\_async(queue,^{

NSLog(@“2”)；

dispatch\_sync(queue,^{

NSLog(@“3”)；

})

NSLog(@“4”)；

})

NSLog(“5”);

结果：输出125或152

1. 异步遇到了同步，回到主线程

NSLog(“1”);

dispatch\_async(dispatch\_get\_global\_queue(0,0),^{

NSLog(@“2”)；

dispatch\_sync(dispatch\_get\_main\_queue(),^{

NSLog(@“3”)；

})

NSLog(@“4”)；  
})

NSLog(“5”);

结果：12534或15234

1. 主线程出现了死循环

dispatch\_async(dispatch\_get\_global\_queue(0,0),^{

NSLog(“1”);

dispatch\_sync(dispatch\_get\_main\_queue(),^{

NSLog(“2”);  
})

NSLog(“3”);  
})

NSLog(“4”);

While(1){};

NSLog(“5”);

结果：14或41

1. GCD定时器

GCD定时器不受runLoop约束，比 NSTimer更加准时

实现

@property(nonatomic,strong) dispatch\_source\_t timer;

int count = 0;

dispatch\_queue\_t queue = dispatch\_get\_main\_queue();

self.timer = dispatch\_source\_create(DISPATCH\_SOURCE\_TYPE\_TIMER, 0, 0, queue);

//设置定时器参数

dispatch\_time start = dispatch\_time(DISPATCH\_TIME\_NOW,(int64\_t)1.0\* NSEC\_PER\_SEC)比当前时间晚1秒

uint64\_t interval = (uint64\_t)(1.0\*NSEC\_PER\_SEC);

dispatch\_source\_set\_timer(self.timer, start, interval, 0);

//设置回调

dispatch\_source\_set\_event\_handler(self.timer, ^{

NSLog(@”fdasf%@”,[NSThread currentThread]);

count ++;

if(count == 4){

dispatch\_cancel(self.timer);

self.timer = nil;

}

});

dispatch\_resume(self.timer);

//只执行一次

double delayInSeconds = 2.0;

dispatch\_time start = dispatch\_time(DISPATCH\_TIME\_NOW,(int64\_t) delayInSeconds \* NSEC\_PER\_SEC)比当前时间晚delayInSeconds秒

dispatch\_after(start, dispatch\_get\_main\_queue(), ^{

//执行事件  
})

1. 运行期间只执行一次代码

static dispatch\_once\_t onceToken;

dispatch\_once(&onceToken,^{

});