

网络安全一个大写的反“作死”
Android开发120天完成9个APP项目
微软企业级内训课程免费学
2016年备战软考-重难点解析

移动开发

首页

Android

iOS

Windows Phone

BlackBerry

webOS

Symbian

bada

OPhone

其他

请输入关键字

搜索

您所在的位置： 移动开发 > 热门推荐 > iOS多线程编程指南(二)线程管理

iOS多线程编程指南(二)线程管理

2013-07-16 10:57 佚名 dreamingwish 字号：T | T

收藏 +

多线程会占用你应用程序(和系统的)的内存使用 and 性能方面的资源。每个线程都需要分配一定的内核内存和应用程序内存空间的内存。管理你的线程和协调其调度所需的核心数据结构存储在使用Wired Memory的内核里面。你线程的堆栈空间和每个线程的数据都被存储在你应用程序的内存空间里面。

AD： 51CTO 网+ 第十二期沙龙：大话数据之美_如何用数据驱动用户体验

线程管理

Mac OS X和iOS里面的每个进程都是有一个或多个线程构成，每个线程都代表一个代码的执行路径。每个应用程序启动时候都是一个线程，它执行程序的主函数。应用程序可以生成额外的线程，其中每个线程执行一个特定功能的代码。

当应用程序生成一个新的线程的时候，该线程变成应用程序进程空间内的一个实体。每个线程都拥有它自己的执行堆栈，由内核调度独立的运行时间片。一个线程可以和其他线程或其他进程通信，执行I/O操作，甚至执行任何你想要它完成的任务。因为它们处于相同的进程空间，所以一个独立应用程序里面的所有线程共享相同的虚拟内存空间，并且具有和进程相同的访问权限。

本章提供了Mac OS X和iOS上面可用线程技术的预览，并给出了如何在你的应用程序里面使用它们的例子。

注意：获取关于Mac OS上面线程架构，或者更多关于线程的背景资料。请参阅技术说明TN2028 - “线程架构”。

1. 线程成本

多线程会占用你应用程序(和系统的)的内存使用 and 性能方面的资源。每个线程都需要分配一定的内核内存和应用程序内存空间的内存。管理你的线程和协调其调度所需的核心数据结构存储在使用Wired Memory的内核里面。你线程的堆栈空间和每个线程的数据都被存储在你应用程序的内存空间里面。这些数据结构里面的大部分都是当你首次创建线程或者进程的时候被创建和初始化的，它们所需的代价成本很高，因为需要和内核交互。

表2-1量化了在你应用程序创建一个新的用户级线程所需的大致成本。这些成本里面的部分是可配置的，比如为辅助线程分配堆栈空间的大小。创建一个线程所需的时间成本是粗略估计的，仅用于当互相比较的时候。线程创建时间很大程度依赖于处理器的负载，计算速度，和可用的系统和程序空间。

Table 2-1 Thread creation costs

Item	Approximate cost	Notes



关注有礼
51CTO官方微信
weixin51cto



3万免费IT视频课程
随时随地学习



专题 Android Studio上手攻略



既然强大的Android Studio来了，有什么理由不去用呢？

iOS新语言swift上手指南
MIUI 6 测评：细节的量变

文章排行

24小时

本周

本月

21个免费的UI界面设计工具、资源及网站
在Eclipse下搭建Android开发环境教程
人人都是开发者：5款傻瓜式APP开发工具
三大移动Web开发框架哪个适合你？
哪个市场最挣钱，腾讯告诉你！
阿里：这是全世界最大的打假团队
Android应当从iOS处窃取的五项最佳功能
iOS 8出色的跨应用通信效果：解读Actio
看库克如何干掉Android
为什么说2015年将是微服务架构元年？

热点职位

更多>>

移动客户端研发工程师
全职/1-3年/本科 12k-25k 多盟

SDK开发工程师
全职/1-3年/大专 7k-10k 游众游戏

Kernel data structures	Approximately 1 KB	This memory is used to store the thread data structures and attributes, much of which is allocated as wired memory and therefore cannot be paged to disk.
Stack space	512 KB (secondary threads) 8 MB (Mac OS X main thread) 1 MB (iOS main thread)	The minimum allowed stack size for secondary threads is 16 KB and the stack size must be a multiple of 4 KB. The space for this memory is set aside in your process space at thread creation time, but the actual pages associated with that memory are not created until they are needed.
Creation time	Approximately 90 microseconds	This value reflects the time between the initial call to create the thread and the time at which the thread's entry point routine began executing. The figures were determined by analyzing the mean and median values generated during thread creation on an Intel-based iMac with a 2 GHz Core Duo processor and 1 GB of RAM running Mac OS X v10.5.

注意:因为底层内核的支持，操作对象(Operation objectis)可能创建线程更快。它们使用内核里面常驻线程池里面的线程来节省创建的时间，而不是每次都创建新的线程。关于更多使用操作对象(Operation objects)的信息，参阅并发编程指南(Concurrency Programming Guide)。

当编写线程代码时另外一个需要考虑的成本是生产成本。设计一个线程应用程序有时会需要根本性改变你应用程序数据结构的组织方式。要做这些改变可能需要避免使用同步，因为本身设计不好的应用可能会造成巨大的性能损失。设计这些数据结构和在线程代码里面调试问题会增加

APP开发工程师

全职/1-3年/本科10k-20k浙江长龙航空

Android软件工程师

全职/3-5年/大专6k-10k广州太乙科技

安卓工程师

全职/3-5年/大专4k-8k索引教育

热点专题

更多>>



Web App开发最佳实践

Web App开发中会面临越来越“重”的问题，如果在开始



Android开发常见“疑

作为Android开发者，最头疼是什么？相信大家会异口同



8大APP给你完美情人节

七夕，是让人听起来就觉得美好的日子，牛郎织女鹊桥相

热点标签

iOS开发 Android开发 Symbian开发 MeeGo开发 BlackBerry开发 Windows Phone开发 Phone Club

Android 4.0 webOS 3.0 智能手机 软件下载

点击这里查看样刊

立即订阅

全站热点



说说HCC2013的那些事儿



《开发月刊》2013年10月刊发布

17次史诗级的 Windows 自动更新崩溃

MemBlaze闪存加速卡PBlaze3的极致新

从零到整：.Net编程结构知识汇总

移动应用App市场营销推广策略

iOS开发之常见疑难问题解决方案大荟

读书



非常网管——网络服务

本书使用通俗易懂的语言，通过大量的实例，从实际应用的角度出发，全面系统地介绍了网络服务操作系统平台、电子邮件系统、Web站

开发一个线程应用所需的时间。然而避免这些消耗的话，可能在运行时候带来更大的问题，如果你的多线程花费太多的时间在锁的等待而没有做任何事情。

2. 创建一个线程

创建低级别的线程相对简单。在所有情况下，你必须有一个函数或方法作为线程的主入口点，你必须使用一个可用的线程例程启动你的线程。以下几个部分介绍了比较常用线程创建的基本线程技术。线程创建使用了这些技术的继承属性的默认设置，由你所使用的技术来决定。关于更多如何配置你的线程的信息，参阅“线程属性配置”部分。

2.1 使用NSThread

使用NSThread来创建线程有两个可以的方法：

使用detachNewThreadSelector:toTarget:withObject:方法来生成一个新的线程。

创建一个新的NSThread对象，并调用它的start方法。（仅在iOS和Mac OS X v10.5及其之后才支持）

这两种创建线程的技术都在你的应用程序里面新建了一个脱离的线程。一个脱离的线程意味着当线程退出的时候线程的资源由系统自动回收。这也同样意味着之后不需要在其他线程里面显示式的连接（join）。因为detachNewThreadSelector:toTarget:withObject:方法在Mac OS X的任何版本都支持，所以在Cocoa应用里面使用多线程的地方经常可以发现它。为了生成一个新的线程，你只要简单的提供你想要使用为线程主体入口的方法的名称（被指定为一个selector），和任何你想在启动时传递给线程的数据。下面的示例演示了这种方法的基本调用，来使用当前对象的自定义方法来生成一个线程。

```
1.  [NSThread detachNewThreadSelector:@selector(myThreadMainMethod:) toTarget:self withObject:nil];
```

在Mac OS X v10.5之前，你使用NSThread类来生成多线程。虽然你可以获取一个NSThread对象并访问线程的属性，但你只能在线程运行之后在其内部做到这些。在Mac OS X v10.5支持创建一个NSThread对象，而无需立即生成一个相应的新线程（这些在iOS里面同样可用）。新版支持使得在线程启动之前获取并设置线程的很多属性成为可能。这也让用线程对象来引用正在运行的线程成为可能。

在Mac OS X v10.5及其之后初始化一个NSThread对象的简单方法是使用initWithTarget:selector:object:方法。该方法和detachNewThreadSelector:toTarget:withObject:方法来初始化一个新的NSThread实例需要相同的额外开销。然而它并没有启动一个线程。为了启动一个线程，你可以显式调用先对象的start方法，如下面代码：

```
1.  NSThread* myThread = [[NSThread alloc] initWithTarget:self
2.  selector:@selector(myThreadMainMethod:)
3.  object:nil];
4.  [myThread start]; // Actually create the thread
```

注意：使用initWithTarget:selector:object:方法的替代办法是子类化NSThread，并重写它的主方法。你可以使用你重写的该方法的版本来实现你线程的主体入口。更多信息，请参阅NSThread Class Reference里面子类化的提示。

如果你拥有一个NSThread对象，它的线程当前真正运行，你可以给该线程发送消息的唯一方法是在你应用程序里面的任何对象使用performSelector:onThread:withObject:waitUntilDone:方法。在Mac OS X v10.5支持在多线程上面执行selectors（而不是在主线程里面），并且它是实现线程间通信的便捷方法。你使用该技术时所发送的消息会被其他线程作为run-loop主体的一部

PHP和MySQL Web开发(原书第3版)

Linux标准教程

XML基础教程

JSP应用开发详解(第三版)

博文推荐

更多>>



nowpaper

Windows
Phone专家



himi

Android开
发专家

利用ISA2006封杀QQ2010

利用word2010中的“邮件”功能批量发

DNS服务器如此简单，35秒轻松搞定！

打造一个属于自己的应用服务自动监控

最新热帖

更多>>

价值上万的SQL学习视频【初学者的福

一道局域网布线的基础题目

【JAVA EE企业级开发四步走完全攻略

通用WAP网站生成系统(PowerDiamond)

八类大学毕业生求职时不受欢迎

分直接执行（当然这些意味着目标线程必须在它的run loop里面运行，参阅“Run Loops”）。当你使用该方法来实现线程通信的时候，你可能仍然需要一个同步操作，但是这比在线程间设置通信端口简单多了。

注意:虽然在线程间的偶尔通信的时候使用该方法很好，但是你不能周期的或频繁的使用performSelector:onThread:withObject:waitUntilDone:来实现线程间的通信。

关于线程间通信的可选方法，参阅“设置线程的脱离状态”部分。

2.2 使用POSIX的多线程

Mac OS X和iOS提供基于C语言支持的使用POSIX线程API来创建线程的方法。该技术实际上可以被任何类型的应用程序使用（包括Cocoa和Cocoa Touch的应用程序），并且如果你当前真为多平台开发应用的话，该技术可能更加方便。你使用来创建线程的POSIX例程被调用的时候，使用pthread_create刚好足够。

列表2-1显示了两个使用POSIX来创建线程的自定义函数。LaunchThread函数创建了一个新的线程，该线程的例程由PosixThreadMainRoutine函数来实现。因为POSIX创建的线程默认情况是可连接的(joinable), 下面的例子改变线程的属性来创建一个脱离的线程。把线程标记为脱离的，当它退出的时候让系统有机会立即回收该线程的资源。

Listing 2-1 Creating a thread in C

```
1.  #include <assert.h>
2.  #include <pthread.h>
3.
4.  void* PosixThreadMainRoutine(void* data)
5.  {
6.      // Do some work here.
7.      return NULL;
8.  }
9.
10. void LaunchThread()
11. {
12.     // Create the thread using POSIX routines.
13.     pthread_attr_t attr;
14.     pthread_t      posixThreadID;
15.     int            returnVal;
16.     returnVal = pthread_attr_init(&attr);
17.     assert(!returnVal);
18.     returnVal = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED)
19.     ;
20.     assert(!returnVal);
21.     int threadError = pthread_create(&posixThreadID, &attr, &PosixThrea
22.     dMainRoutine, NULL);
23.     returnVal = pthread_attr_destroy(&attr);
24.     assert(!returnVal);
25.     if (threadError != 0)
26.     {
27.         // Report an error.
```

如果你把上面列表的代码添加到你任何一个源文件，并且调用LaunchThread函数，它将会在你的应用程序里面创建一个新的脱离线程。当然，新创建的线程使用该代码没有做任何有用的事情。线程将会加载并立即退出。为了让它更有兴趣，你需要添加代码到PosixThreadMainRoutine函数里面来做一些实际的工作。为了保证线程知道该干什么，你可以在创建的时候给线程传递一个数据的指针。把该指针作为pthread_create的最后一个参数。

为了在新建的线程里面和你应用程序的主线程通信，你需要建立一条和目标线程之间的稳定的通信路径。对于基于C语言的应用程序，有几种办法来实现线程间的通信，包括使用端口

(ports)，条件(conditions)和共享内存(shared memory)。对于长期存在的线程，你应该几乎总是成立某种线程间的通信机制，让你的应用程序的主线程有办法来检查线程的状态或在应用程序退出时干净关闭它。

关于更多介绍POSIX线程函数的信息，参阅pthread的主页。

2.3 使用NSObject来生成一个线程

在iOS和Mac OS X v10.5及其之后，所有的对象都可能生成一个新的线程，并用它来执行它任意的的方法。方法performSelectorInBackground:withObject:新生成一个脱离的线程，使用指定的方法作为新线程的主体入口点。比如，如果你有一些对象（使用变量myObj来代表），并且这些对象拥有一个你想在后台运行的doSomething的方法，你可以使用如下的代码来生成一个新的线程：

```
1. [myObj performSelectorInBackground:@selector(doSomething) withObject:nil];
```

调用该方法的效果和你在当前对象里面使用NSThread的detachNewThreadSelector:toTarget:withObject:传递selectore，object作为参数的方法一样。新的线程将会被立即生成并运行，它使用默认的设置。在selectore内部，你必须配置线程就像你在任何线程里面一样。比如，你可能需要设置一个自动释放池（如果你没有使用垃圾回收机制），在你要使用它的时候配置线程的run loop。关于更是介绍如果配置线程的信息，参阅“配置线程属性”部分。

2.4 使用其他线程技术

尽管POSIX例程和NSThread类被推荐使用来创建低级线程，但是其他基于C语言的技术在Mac OS X上面同样可用。在这其中，唯一一个可以考虑使用的是多处理服务（Multiprocessing Services），它本身就是在POSIX线程上执行。多处理服务是专门为早期的Mac OS版本开发的，后来在Mac OS X里面的Carbon应用程序上面同样适用。如果你有代码真是有该技术，你可以继续使用它，尽管你应该把这些代码转化为POSIX。该技术在iOS上面不可用。

关于更多如何使用多处理服务的信息，参阅多处理服务编程指南（Multiprocessing Services Programming Guide）。

别走开，下页内容更精彩

2.5 在Cocoa程序上面使用POSIX线程

尽管NSThread类是Cocoa应用程序里面创建多线程的主要接口，如果可以更方便的话你可以任意使用POSIX线程带替代。例如，如果你的代码里面已经使用了它，而你又不想改写它的话，这时你可能需要使用POSIX多线程。如果你真打算在Cocoa程序里面使用POSIX线程，你应该了解如果在Cocoa和线程间交互，并遵循以下部分的一些指南。

Cocoa框架的保护

对于多线程的应用程序，Cocoa框架使用锁和其他同步方式来保证代码的正确执行。为了保护这些锁造成在单线程里面性能的损失，Cocoa直到应用程序使用NSThread类生成它的第一个新的线程的时候才创建这些锁。如果你仅且使用POSIX例程来生成新的线程，Cocoa不会收到关于你的应用程序当前变为多线程的通知。当这些刚好发生的时候，涉及Cocoa框架的操作可能会破坏甚至让你的应用程序崩溃。

为了让Cocoa知道你正打算使用多线程，你所需要做的是使用NSThread类生成一个线程，并让它立即退出。你线程的主体入口点不需要做任何事情。只需要使用NSThread来生成一个线程就足够保证Cocoa框架所需的锁到位。

如果你不确定Cocoa是否已经知道你的程序是多线程的，你可以使用NSThread的isMultiThreaded方法来检验一下。

混合POSIX和Cocoa的锁

在同一个应用程序里面混合使用POSIX和Cocoa的锁很安全。Cocoa锁和条件对象基本上只是封装了POSIX的互斥体和条件。然而给定一个锁，你必须总是使用同样的接口来创建和操纵该锁。换言之，你不能使用Cocoa的NSLock对象来操纵一个你使用pthread_mutex_init函数生成的互斥体，反之亦然。

3. 配置线程属性

创建线程之后，或者有时候是之前，你可能需要配置不同的线程环境。以下部分描述了一些你可以做的改变，和在什么时候你需要做这些改变。

3.1 配置线程的堆栈大小

对于每个你新创建的线程，系统会在你的进程空间里面分配一定的内存作为该线程的堆栈。该堆栈管理堆栈帧，也是任何线程局部变量声明的地方。给线程分配的内存大小在“线程成本”里面已经列举了。

如果你想要改变一个给定线程的堆栈大小，你必须在创建该线程之前做一些操作。所有的线程技术提供了一些办法来设置线程堆栈的大小。虽然可以使用NSThread来设置堆栈大小，但是它只能在iOS和Mac OS X v10.5及其之后才可用。表2-2列出了每种技术的对于不同的操作。

Table 2-2 Setting the stack size of a thread

Technology	Option
Cocoa	In iOS and Mac OS X v10.5 and later, allocate and initialize an NSThread object (do not use thedetachNewThreadSelector:toTarget:withObject: method). Before calling the start method of the thread object, use thesetStackSize: method to specify the new stack size.
POSIX	Create a new pthread_attr_t structure and use the pthread_attr_setstacksize function to change the default stack size. Pass the attributes to the pthread_create function when creating your thread.
Multiprocessing Services	Pass the appropriate stack size value to the MPCreateTask function when you create your thread.

3.2 配置线程本地存储

每个线程都维护了一个键-值的字典，它可以在线程里面的任何地方被访问。你可以使用该字典来保存一些信息，这些信息在整个线程的执行过程中都保持不变。比如，你可以使用它来存储在你的整个线程过程中Run loop里面多次迭代的状态信息。

Cocoa和POSIX以不同的方式保存线程的字典，所以你不能混淆并同时调用者两种技术。然而只要你在你的线程代码里面坚持使用了其中一种技术，最终的结果应该是一样的。在Cocoa里面，你使用NSThread的threadDictionary方法来检索一个NSMutableDictionary对象，你可以在它里面

添加任何线程需要的键。在POSIX里面，你使用pthread_setspecific和pthread_getspecific函数来设置和访问你线程的键和值。

3.3 设置线程的脱离状态

大部分上层的线程技术都默认创建了**脱离线程**（Detached thread）。大部分情况下，**脱离线程**（Detached thread）更受欢迎，因为它们允许系统在线程完成的时候立即释放它的数据结构。**脱离线程**同时不需要显示的和你的应用程序交互。意味着线程检索的结果由你来决定。相比之下，系统不回收**可连接线程**（Joinable thread）的资源直到另一个线程明确加入该线程，这个过程可能会阻止线程执行加入。

你可以认为**可连接线程**类似于子线程。虽然你作为独立线程运行，但是**可连接线程**在它资源可以被系统回收之前必须被其他线程连接。**可连接线程**同时提供了一个显示的方式来把数据从一个正在退出的线程传递到其他线程。在它退出之前，**可连接线程**可以传递一个数据指针或者其他返回给pthread_exit函数。其他线程可以通过pthread_join函数来拿到这些数据。

重要：在应用程序退出时，脱离线程可以立即被中断，而可连接线程则不可以。每个可连接线程必须在进程被允许可以退出的时候被连接。所以当线程处于周期性工作而不允许被中断的时候，比如保存数据到硬盘，可连接线程是最佳选择。

如果你想要创建可连接线程，唯一的办法是使用POSIX线程。POSIX默认创建的线程是可连接的。为了把线程标记为脱离的或可连接的，使用pthread_attr_setdetachstate函数来修改正在创建的线程的属性。在线程启动后，你可以通过调用pthread_detach函数来把线程修改为可连接的。关于更多POSIX线程函数信息，参与pthread主页。关于更多如果连接一个线程，参阅pthread_join的主页。

3.4 设置线程的优先级

你创建的任何线程默认的优先级是和你本身线程相同。内核调度算法在决定该运行那个线程时，把线程的优先级作为考量因素，较高优先级的线程会比较低优先级的线程具有更多的运行机会。较高优先级不保证你的线程具体执行的时间，只是相比较低优先级的线程，它更有可能被调度器选择执行而已。

重要：让你的线程处于默认优先级值是一个不错的选择。增加某些线程的优先级，同时有可能增加了某些较低优先级线程的饥饿程度。如果你的应用程序包含较高优先级和较低优先级线程，而且它们之间必须交互，那么较低优先级的饥饿状态有可能阻塞其他线程，并造成性能瓶颈。

如果你想改变线程的优先级，Cocoa和POSIX都提供了一种方法来实现。对于Cocoa线程而言，你可以使用NSThread的setThreadPriority:类方法来设置当前运行线程的优先级。对于POSIX线程，你可以使用pthread_setschedparam函数来实现。关于更多信息，参与NSThread Class Reference或pthread_setschedparam主页。

4. 编写你线程的主体入口点

对于大部分而言，Mac OS X上面线程结构的主体入口点和其他平台基本一样。你需要初始化你的数据结构，做一些工作或可行的设置一个run loop，并在线程代码被执行完后清理它。根据设计，当你写的主体入口点的时候有可能需要采取一些额外的步骤。

4.1 创建一个自动释放池（Autorelease Pool）

在Objective - C框架链接的应用程序，通常在它们的每一个线程必须创建至少一个自动释放池。如果应用程序使用管理模型，即应用程序处理的retain和release对象，那么自动释放池捕获任何从该线程autorelease的对象。

如果应用程序使用的垃圾回收机制，而不是管理的内存模型，那么创建一个自动释放池不是绝对必要的。在垃圾回收的应用程序里面，一个自动释放池是无害的，而且大部分情况是被忽略。允许通过个代码管理必须同时支持垃圾回收和内存管理模型。在这种情况下，内存管理模型必须支持自动释放池，当应用程序运行垃圾回收的时候，自动释放池只是被忽略而已。

如果你的应用程序使用内存管理模型，在你编写线程主体入口的时候第一件事情就是创建一个自动释放池。同样，在你的线程最后应该销毁该自动释放池。该池保证自动释放。虽然对象被调用，但是它们不被release直到线程退出。列表2-2显示了线程主体入口使用自动释放池的基本结构。

Listing 2-2 Defining your thread entry point routine

```
1.  - (void)myThreadMainRoutine
2.  {
3.      NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init]; // Top-level pool
4.      // Do thread work here.
5.      [pool release]; // Release the objects in the pool.
6.  }
```

因为高级的自动释放池不会释放它的对象直到线程退出。长时运行的线程需求新建额外的自动释放池来更频繁的释放它的对象。比如，一个使用run loop的线程可能在每次运行完一次循环的时候创建并释放该自动释放池。更频繁的释放对象可以防止你的应用程序内存占用太大造成性能问题。虽然对于任何与性能相关的行为，你应该测量你代码的实际表现，并适当地调整使用自动释放池。

关于更多内存管理的信息和自动释放池，参阅“内存高级管理编程指南(Advanced Memory Management Programming Guide)”。

4.2 设置异常处理

如果你的应用程序捕获并处理异常，那么你的线程代码应该时刻准备捕获任何可能发生的异常。虽然最好的办法是在异常发生的地方捕获并处理它，但是如果在你的线程里面捕获一个抛出的异常失败的话有可能造成你的应用程序强退。在你线程的主体入口点安装一个try/catch模块，可以让你捕获任何未知的异常，并提供一个合适的响应。

当在Xcode构建你项目的时候，你可以使用C++或者Objective-C的异常处理风格。关于更多设置如何在Objective-C里面抛出和捕获异常的信息，参阅Exception Programming Topics。

4.3 设置一个Run Loop

当你想编写一个独立运行的线程时，你有两种选择。第一种选择是写代码作为一个长期的任务，很少甚至不中断，线程完成的时候退出。第二种选择是把你的线程放入一个循环里面，让它动态的处理到来的任务请求。第一种方法不需要在你的代码指定任何东西；你只需要启动的时候做你打算做的事情即可。然而第二种选择需要在你的线程里面添加一个run loop。

Mac OS X和iOS提供了在每个线程实现run loop内置支持。Cocoa、Carbon和UIKit自动在你应用程序的主线程启动一个run loop，但是如果你创建任何辅助线程，你必须手工的设置一个run loop并启动它。

关于更多使用和配置run loop的信息，参阅“Run Loops”部分。

5. 中断线程

退出一个线程推荐的方法是让它在它主体入口点正常退出。经管Cocoa、POSIX和Multiprocessing Services提供了直接杀死线程的例程，但是使用这些例程是强烈不鼓励的。杀

死一个线程阻止了线程本身的清理工作。线程分配的内存可能造成泄露，并且其他线程当前使用的资源可能没有被正确清理干净，之后造成潜在的问题。

如果你的应用程序需要在一个操作中间中断一个线程，你应该设计你的线程响应取消或退出的消息。对于长时运行的操作，这意味着周期性停止工作来检查该消息是否到来。如果该消息的确到来并要求线程退出，那么线程就有机会来执行任何清理和退出工作；否则，它返回继续工作和处理下一个数据块。

响应取消消息的一个方法是使用run loop的输入源来接收这些消息。列表2-3显示了该结构的类似代码在你的线程的主体入口里面是怎么样的（该示例显示了主循环部分，不包括设立一个自动释放池或配置实际的工作步骤）。该示例在run loop上面安装了一个自定义的输入源，它可以从此线程接收消息。关于更多设置输入源的信息，参阅“配置Run Loop源”。执行工作的总和的一部分后，线程运行的run loop来查看是否有消息抵达输入源。如果没有，run loop立即退出，并且循环继续处理下一个数据块。因为该处理器并没有直接的访问exitNow局部变量，退出条件是通过线程的字典来传输的。

Listing 2-3 Checking for an exit condition during a long job

```
1.  - (void)threadMainRoutine
2.  {
3.      BOOL moreWorkToDo = YES;
4.      BOOL exitNow = NO;
5.      NSRunLoop* runLoop = [NSRunLoop currentRunLoop];
6.
7.      // Add the exitNow BOOL to the thread dictionary.
8.      NSMutableDictionary* threadDict = [[NSThread currentThread] threadDictionary];
9.      [threadDict setValue:[NSNumber numberWithInt:exitNow] forKey:@"ThreadShouldExitNow"];
10.
11.     // Install an input source.
12.     [self myInstallCustomInputSource];
13.
14.     while (moreWorkToDo && !exitNow)
15.     {
16.         // Do one chunk of a larger body of work here.
17.         // Change the value of the moreWorkToDo Boolean when done.
18.
19.         // Run the run loop but timeout immediately if the input source isn't waiting to fire.
20.         [runLoop runUntilDate:[NSDate date]];
21.
22.         // Check to see if an input source handler changed the exitNow value.
23.         exitNow = [[threadDict valueForKey:@"ThreadShouldExitNow"] boolValue];
24.     }
25. }
```

【编辑推荐】

1. [GCD使用攻略](#)
2. [iOS多线程编程知多少](#)
3. [GCD实战一:使用串行队列实现简单的预加载](#)
4. [GCD实战二:资源竞争](#)
5. [iOS多线程编程指南\(一\)关于多线程编程](#)

【责任编辑：milk TEL：（010）68476606】

原文：[iOS多线程编程指南\(二\)线程管理](#)

[返回移动开发首页](#)



微信扫一扫

移动首页版

同时，您也可以在移动端浏览器上输入“www.51cto.com”随时随地浏览和分享最具价值的技术内容

优质技术内容尽收掌中

微博推荐



51CTO移动开
51CTO移
动开发频道



51CTO官方微
51cto官方
微博



51CTO技术博
51CTO技
术博客官方



51CTO技术社
51CTO技
术社区(ho



51CTO熊平
51CTO传
媒总裁熊平



小林51CTO
北京无忧创
想信息技术

一键关注

[注册微博](#)

分享到：

0

[收藏](#) | [打印](#) | [复制](#)



给力
(0票)



动心
(0票)



废话
(0票)



专业
(0票)




标题党
(0票)



路过
(0票)

0 条评论，0 人参与。

 0



我有话说...

使用社交帐号登录

发布前先点击左边的按钮登录

最新评论

还没有评论

友言？

关于 [iOS多线程](#) [多线程概念](#) [多线程入门](#) [线程管理](#) 的更多文章

iOS多线程编程指南（拓展篇）
iOS多线程编程指南(四)线程同步
iOS多线程编程指南(三)Run Loop

全面掌握iOS多线程攻略

多线程是一个比较轻量级的方法来实现单个应用程序内多个代码执行[详细]

iOS多线程编程指南(二)线程管理
iOS多线程编程指南(一)关于多线程编程



栏目热门

[更多>>](#)

100分的输家：一个146年历史的诺基亚为何4
李开复：移动互联网时代该如何创业
开源免费天气预报接口API以及全国所有地区
三星或将推“土豪金”Galaxy S4 死磕iPhone 5
微软收购诺基亚：移动开发者应该看到什么

同期最新

[更多>>](#)

苹果将继续扁平化之路，技术支持网站使用新
游戏开发者外包美术工作的注意事项
iOS多线程编程指南(一)关于多线程编程
写给初级游戏设计师的12条建议
支付宝的超级App野心：首屏App设置 打通支

移动开发 频道导航

平台

[移动Web](#) | [Android](#) | [iOS](#) | [Windows Phone](#)

应用

[移动应用](#) | [移动团队](#) | [应用商店](#) | [专题汇总](#) | [Phone Club](#)

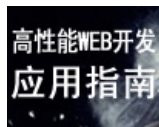
观察

[业界观察](#) | [调查数据](#) | [移动信息化](#)

Android

[热点](#) | [资讯](#) | [基础](#) | [多媒体](#) | [数据库](#) | [设计](#) | [工具](#) | [编译](#)

热点推荐

[Android开发应用详解](#)[那些性感的让人尖叫
的程序员](#)[HTML5 下一代Web开发
标准详解](#)[高性能WEB开发应用指
南](#)[Ubuntu开源技术交流
频道](#)

热门标签: [windows频道](#) [移动开发](#) [云计算](#) [objective-c](#) [tp-link路由器设置图解](#) [html5](#)



上百度公益开放平台 为您的公益组织免费推广吧!

51CTO旗下网站

领先的IT技术网站 51CTO 领先的中文存储媒体 WatchStor 中国首个CIO网站 CIOage 中国首家数字医疗网站 HC3i

Copyright©2005-2016 51CTO.COM 版权所有 未经许可 请勿转载