Atitit.并发编程原理与概论 attilax总结

[1. 并发一般涉及如下几个方面： 2](#_Toc28261)

[2. 线程安全性 （　 　　2.2　原子性　　　　2.3　加锁机制 2](#_Toc9967)

[2.1. 线程封闭　　　3.3.1　Ad-hoc线程封闭　　　3.3.2　栈封闭　　　3.3.3　ThreadLocal类 2](#_Toc16642)

[3. 异步 2](#_Toc23907)

[4. 同步与锁关键字 2](#_Toc26465)

[5. 5.2　并发容器与并发集合 2](#_Toc3897)

[6. Future模式 2](#_Toc23720)

[7. 5.3　阻塞队列和生产者-消费者模式（　　　5.3.2　串行线程封闭 　　　5.3.3　双端队列与工作密取 2](#_Toc8741)

[8. 5.4　阻塞方法与中断方法 2](#_Toc4674)

[9. 5.5　同步工具类 　　　5.5.1　闭锁 　　　5.5.2　FutureTask　　　5.5.3　信号量 　　　5.5.4　栅栏 3](#_Toc25186)

[10. 5.6　构建高效且可伸缩的结果缓存 3](#_Toc10763)

[11. 线程池 3](#_Toc7846)

[12. 7.1　任务取消 3](#_Toc17159)

[13. 任务并行库(TPL) 4](#_Toc24698)

[14. 死锁的避免与诊断 4](#_Toc29896)

[15. 原子变量与非阻塞同步机制 4](#_Toc3227)

[16. 协程 4](#_Toc14046)

[17.](#_Toc21259) **[异步、多线程、任务、并行的本质](#_Toc21259)** [4](#_Toc21259)

[18.](#_Toc7444) **[现在，该用什么来编写多线程](#_Toc7444)**[5](#_Toc7444)

[18.1.1. 1，异步编程 5](#_Toc21751)

[19. 响应式编程 6](#_Toc22085)

[20. ，数据流编程 6](#_Toc22186)

[20.1. 5，Actor模型 7](#_Toc2817)

[21. Qa 7](#_Toc17943)

[22. Java c# .net c++的并发技术 7](#_Toc25199)

[22.1. Java并发编程实战（第16届Jolt大奖提名图书，Java并发编程必读佳作 8](#_Toc25291)

[22.2. Java并发技术 8](#_Toc26366)

[22.3. 《C#多线程编程实战( (（美...【 8](#_Toc31293)

[22.4. Line 278: 　　《C++并发编程实战》 Line 285: 第1章 你好，C++并发世界 9](#_Toc4585)

[22.5. 《C#并发编程经典实例》 9](#_Toc986)

# 并发一般涉及如下几个方面：

1. 多线程编程（已过时，不介绍）
2. 异步编程
3. 并行编程
4. 响应式编程
5. 数据流编程

# 线程安全性 （　 　　2.2　原子性　　　　2.3　加锁机制

## 线程封闭　　　3.3.1　Ad-hoc线程封闭　　　3.3.2　栈封闭　　　3.3.3　ThreadLocal类

# 异步

# 同步与锁关键字

# 5.2　并发容器与并发集合

# Future模式

作者:: 绰号:老哇的爪子 （ 全名：：Attilax Akbar Al Rapanui 阿提拉克斯 阿克巴 阿尔 拉帕努伊 ）

汉字名：艾提拉（艾龙），   EMAIL:1466519819@qq.com

转载请注明来源： http://blog.csdn.net/attilax

# 5.3　阻塞队列和生产者-消费者模式（　　　5.3.2　串行线程封闭 　　　5.3.3　双端队列与工作密取

# 5.4　阻塞方法与中断方法

# 5.5　同步工具类 　　　5.5.1　闭锁 　　　5.5.2　FutureTask　　　5.5.3　信号量 　　　5.5.4　栅栏

# 5.6　构建高效且可伸缩的结果缓存

# 线程池

1. 线程池的使用  
   　　8.1　在任务与执行策略之间的隐性耦合  
   　　　8.1.1　线程饥饿死锁  
   　　　8.1.2　运行时间较长的任务  
   　　8.2　设置线程池的大小  
   　　8.3　配置ThreadPoolExecutor  
   　　　8.3.1　线程的创建与销毁  
   　　　8.3.2　管理队列任务  
   　　　8.3.3　饱和策略  
   　　　8.3.4　线程工厂

# 7.1　任务取消

　　　7.1.1　中断  
　　　7.1.2　中断策略  
　　　7.1.3　响应中断  
　　　7.1.4　示例：计时运行  
　　　7.1.5　通过Future来实现取消  
　　　7.1.6　处理不可中断的阻塞  
　　　7.1.7　采用newTaskFor来封装非标准的取消  
　　7.2　停止基于线程的服务  
　　　7.2.1　示例：日志服务  
　　　7.2.2　关闭ExecutorService  
　　　7.2.3　“毒丸”对象  
　　　7.2.4　示例：只执行一次的服务  
　　　7.2.5　shutdownNow的局限性  
　　7.3　处理非正常的线程终止

# 任务并行库(TPL)

# 死锁的避免与诊断

# 原子变量与非阻塞同步机制

# 协程

# ****异步、多线程、任务、并行的本质****

这四个概念对应在CLR中的本质，本质都是多线程。

异步，简单的讲就是BeginInvoke、EndInvoke模式，它在CLR内部线程池进行管理；

多线程，体现在C#中，可以由类型Thread发起。也可以由ThreadPool发起。前者不受CLR线程池管理，后者则是。FCL团队为了各种编程模型的方便，还另外提供了BackgroundWorker和若干个Timer，基本上它们都是ThreadPool的加强，增加了一些和调用者线程的交互功能；

任务（Task），为FCL4.0新增的功能，在一个称之为任务并行库（TPL）的地方，其实也就是System.Threading.Tasks命名空间下。任务并行库名字取的很玄乎，其实它也是CLR线程池的加强。优化了线程间的调度算法，增加了和调用者线程的交互功能；

并行（Parallel），为FCL4.0新增的功能，也属于TPL。并行在后台使用Task进行管理，说白了，因为Task使用的线程池线程，所以Parallel自然使用的也是线程池线程进行管理，它的本质仅仅是进一步简化了Task。在这里要增进一个对于并行的理解。实际上，多线程天然就是并行的。及时不用任务并行库，用Thread类型新起两个线程，CLR或者说Windows系统也会将这两个线程根据需要安排到两个CPU上去执行。所以，并不是因为多了任务并行库，CLR才支持并行计算，任务并行库只是提供了一组API，使我们能够更好的操纵线程进行并行开发而已。

# ****现在，该用什么来编写多线程****

如果你在FRAMEWORK4.0下编写代码，那么应该按照这个优先级来撰写多线程代码：

|  |  |  |
| --- | --- | --- |
| 优先 | 次优先 | 不得以 |
| Parallel（含扩展库PLinq）  Task | ThreadPool（BackgroundWorker，Timer）  异步 | Thread |

这个表满足了大部分情况下的一个优先级指导，但在某些情况下会有例外。

多线程编程（已过时，不介绍）

### 1，异步编程

异步编程就是使用future模式（又称promise）或者回调机制来实现（Non-blocking on waiting）。

微软还特意把异步编程分作了3种不同的模型：基于任务的模式（TAP）就是我上面推荐的这种，基于事件的模式（EAP）和异步编程模型（APM）我上面不推荐的事件和回调。

# 响应式编程

响应式编程最近成为了一个Buzzword，其实微软6年前就开始给.NET提供一个**Reactive   
Extensions**了。一开始要理解响应式编程有点困难，但是一旦理解了，你就会对它的强大功能爱不释手。简单来说，响应式编程把事件流看作数据流，不过数据流是从IEnumable中拉取的，而数据流是从IObservable推送给你的。为什么响应式编程可以实现并发呢？这是因为Rx做到线程不可知，每次事件触发，后续的处理会从线程池中任意取出一个线程来处理。且可以对事件设置窗口期和限流。举个例子，你可以用Rx来让搜索文本框进行延迟处理（而不用类似我很早的时候用个定时器来延迟了）。

# ，数据流编程

数据流（DataFlow）编程可能大家就更陌生了，不过还是有些常用场景可以使用数据流来解决。数据流其实是在任务并行库（TPL）上衍生出来的一套处理数据的扩展（也结合了异步的特性），TPL也是处理并行编程中任务并行和数据并行的基础库。  
  
望文生义，TPL DataFlow就是对数据进行一连串处理，首先为这样的处理定义一套网格（mesh），网格中可以定义分叉（fork）、连接（join）、循环（loop）。数据流入这样的处理网格就能够并行的被处理。你可以认为网格是一种升级版的管道，实际上很多时候就是被当作管道来使用。使用场景可以是“**分析文本文件中词频**”，也可以是“**处理生产者/消费者问题**”。

## 5，Actor模型

Scala有Akka，其实微软研究院也推出了Orleans来支持了Actor模型的实现，当然也有Akka.NET可用。Orleans设计的目标是为了方便程序员开发需要大规模扩展的云服务,

# Qa

**.2：何时用异步，何时用线程或线程池**

这需要从“IO操作的DMA（Direct Memory Access）模式”讲起。通过DMA的数据交换几乎可以不损耗CPU的资源。在硬件部分，硬盘、网卡、声卡、显卡等都有DMA功能。可以简单的认为，当我们的工作线程需要操作I/O资源的时候（如读取一个大文件、读取一个网页、读取Socke包等），我们就需要用异步去做这些事情。异步模式只会在工作开始以及工作结束的时候占用CLR线程池，其它时候由硬盘、网卡等硬件设备来处理具体的工作，这就不会过多占用到CPU空间和时间损耗。

概括而言：

计算密集型工作，直接采用线程；

IO密集型工作，采用异步机制；

当我们不清楚什么工作是I/O密集型的，一个不是很恰当的指导就是：查看FCL类型成员，如果成员提供了类似BeginDosomething方法的，则优先使用它，而不是新起一个线程或丢到线程池。

**3.4：何时用Thread**  
以上的各种线程模型，它们最终都是Thread。 那么什么时候需要Thread直接出场呢？

最重要的使用Thread的理由是，我们需要控制线程的优先级。Thread之上的线程模型都不支持优先级设置。设置一个线程的高优先级可以使它获得更多的CPU时间；

再者，可以控制线程为前台线程。当然，由Thread新起的线程默认就是前台线程。前台线程不随着调用者线程的中断而中断，这使得我们可以用Thread来进行一些关键性的操作。

# Java c# .net c++的并发技术

## Java并发编程实战（第16届Jolt大奖提名图书，Java并发编程必读佳作

作者:[Brian](http://www.dangdang.com/author/Brian_1" \t "http://product.dangdang.com/_blank) [Goetz](http://www.dangdang.com/author/Goetz_1" \t "http://product.dangdang.com/_blank)，Tim [Peierls](http://www.dangdang.com/author/Peierls_1" \t "http://product.dangdang.com/_blank)，[Joshua](http://www.dangdang.com/author/Joshua_1" \t "http://product.dangdang.com/_blank) [Bloch](http://www.dangdang.com/author/Bloch_1" \t "http://product.dangdang.com/_blank)，[Joseph](http://www.dangdang.com/author/Joseph_1" \t "http://product.dangdang.com/_blank) [Bowbeer](http://www.dangdang.com/author/Bowbeer_1" \t "http://product.dangdang.com/_blank)，[David](http://www.dangdang.com/author/David_1" \t "http://product.dangdang.com/_blank) [Holmes](http://www.dangdang.com/author/Holmes_1" \t "http://product.dangdang.com/_blank)，[Doug](http://www.dangdang.com/author/Doug_1" \t "http://product.dangdang.com/_blank) Lea 著，[童云兰](http://www.dangdang.com/author/%CD%AF%D4%C6%C0%BC_1" \t "http://product.dangdang.com/_blank)　等译

第1章　简介

　第2章　线程安全性

　第3章　对象的共享

　第4章　对象的组合

　第5章　基础构建模块

　第6章　任务执行

　第7章　取消与关闭

　第8章　线程池的使用

　第9章　图形用户界面应用程序

　第10章　避免活跃性危险

　第11章　性能与可伸缩性

　第12章　并发程序的测试

　第13章　显式锁

　第14章　构建自定义的同步工具

　第15章　原子变量与非阻塞同步机制

　第16章　Java内存模型

## Java并发技术

Executor框架

## 《C#多线程编程实战( (（美...【

1. 线程基础 　（ lock关键字 　1.11　使用Monitor类锁定资源  
    第2章　线程同步  
    第3章　使用线程池  
   第4章　使用任务并行库  
   第5章　使用C# 5.0  
   第6章　使用并发集合  
   第7章　使用PLINQ  
   第8章　使用Reactive Extensions  
   第9章　使用异步I/O  
   第10章　并行编程模式  
   第11章　更多信息

## Line 278: 　　《C++并发编程实战》 Line 285: 第1章 你好，C++并发世界

Line 300: 第2章 管理线程

Line 311: 第3章 在线程间共享数据

Line 329: 第4章 同步并发操作

Line 348: 第5章 C++内存模型和原子

Line 370: 第6章 设计基于锁的并发数据结构

Line 380: 第7章 设计无锁的并发数据结构

## 《C#并发编程经典实例》

第1 章　并发编程概述

第2 章　异步编程基础

第3 章　并行开发的基础

第4 章　数据流基础

第5 章　Rx 基础

第6 章　测试技巧

第7 章　互操作

第8 章　集合

第9 章　取消

第10 章　函数式OOP

第11 章　同步

第12 章　调度

第13 章　实用技巧

参考资料

异步、多线程、任务、并行编程之一：选择合适的多线程模型 - Luminji - 博客园.html

我为什么喜欢用C#来做并发编程-博客-云栖社区-阿里云.html

atiend