atitit 高并发之道 attilax著.docx

[1. 概念 2](#_Toc11614)

[2. 并发一般涉及如下几个方面： 4](#_Toc21856)

[2.1. 多线程编程（已过时，不介绍） 4](#_Toc5437)

[2.2. 异步编程 4](#_Toc23855)

[2.3. 并行编程 4](#_Toc8491)

[2.4. 响应式编程 4](#_Toc10863)

[2.5. 数据流编程 4](#_Toc21736)

[3. Atitit 三种并发编程模型 并行工作者 线程模型 流水线（Assembly Line） 事件驱动模型 函数式并发模型 4](#_Toc28893)

[4. 任务并行库(TPL) 5](#_Toc16467)

[5. 定时器（java是线程实现的） 5](#_Toc18339)

[6. 线程 5](#_Toc29065)

[6.1. 线程安全性 （　 　　2.2　原子性　　　　2.3　加锁机制 5](#_Toc4487)

[6.2. 5](#_Toc237)

[6.3. 线程封闭　　　3.3.1　Ad-hoc线程封闭　　　3.3.2　栈封闭　　　3.3.3　ThreadLocal类 5](#_Toc2886)

[7. 线程池 5](#_Toc20109)

[8. 异步 6](#_Toc23029)

[8.1. 异步 6](#_Toc887)

[8.2. 同步与锁关键字 6](#_Toc366)

[8.3. 5.2 6](#_Toc16131)

[9. 并发容器与并发集合 6](#_Toc11016)

[9.1. Future模式 6](#_Toc22299)

[9.2. 6](#_Toc10030)

[10. 其他 6](#_Toc3073)

[10.1. 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 6](#_Toc10387)

[10.2. 5.4　阻塞方法与中断方法 原子变量与非阻塞同步机制 6](#_Toc21296)

[10.3. 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 6](#_Toc28173)

[10.4. 死锁的避免与诊断 7](#_Toc14915)

[10.5. 7](#_Toc30827)

[11. 协程 7](#_Toc10756)

[12. 异步、多线程、任务、并行的本质 7](#_Toc5047)

[13. 现在，该用什么来编写并发程序、多线程  7](#_Toc21270)

[13.1.1. 1，异步编程 8](#_Toc30658)

[13.2. 响应式编程 8](#_Toc6361)

[13.3. ，数据流编程 9](#_Toc8691)

[13.4. 5，Actor模型 9](#_Toc13428)

[14. Qa 9](#_Toc23216)

[15. Java c# .net c++的并发技术 10](#_Toc22829)

[15.1. 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 10](#_Toc10965)

[15.2. Java并发技术 11](#_Toc11377)

[16. 22. Java c# .net c++的并发技术 7 11](#_Toc27155)

[16.1. 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 11](#_Toc14143)

[16.2. 22.2. Java并发技术 8 11](#_Toc5429)

[16.3. 22.3. 《C#多线程编程实战( (（美...【 8 11](#_Toc25558)

[16.4. 22.4. Line 278: 　　《C++并发编程实战》 Line 285: 第1章 你好，C++并发世界 9 11](#_Toc21401)

[16.5. 22.5. 《C#并发编程经典实例》 9 11](#_Toc29134)

[17. 参考资料 12](#_Toc3424)

[17.1. atitit 并发编程原理与概论 attilax总结.docx 12](#_Toc23563)

# 概念

1. 简 介  
   编写正确的程序很难，而编写正确的并发程序則难上加难。与串行程序相比，在并发程序中存在更多容易出错的地方。那么，为什么还要编写并发程序?线程是Java语言中不可或缺的重要功能，它们能使复杂的异步代码变得更简单，从而极大地简化了复杂系统的开发。此外，要想充分发挥多处理器系统的强大计算能力，最简单的方式就是使用线程。随着处理器数量的持续增长，如何高效地使用并发正变得越来越重要。  
   　　1.1 并发简史  
   在早期的计算机中不包含操作系统，它们从头到尾只执行一个程序，并且这个程序能访问计算机中的所有资源。在这种裸机环境中，不仅很难编写和运行程序，而且每次只能运行一个程序，这对于昂贵并且稀有的计算机资源来说也是一种浪费。  
   操作系统的出现使得计算机每次能运行多个程序，并且不同的程序都在单独的进程中运行：操作系统为各个独立执行的进程分配各种资源，包括内存，文件句柄以及安全证书等。如果需要的话，在不同的进程之间可以通过一些粗粒度的通信机制来交换数据，包括：套接字、信号处理器、共享内存、信号量以及文件等。  
   之所以在计算机中加入操作系统来实现多个程序的同时执行，主要是基于以下原因：  
   资源利用率。在某些情况下，程序必须等待某个外部操作执行完成，例如输入操作或输出操作等，而在等待时程序无法执行其他任何工作。因此，如果在等待的同时可以运行另一个程序，那么无疑将提高资源的利用率。  
   公平性。不同的用户和翠序对于计算机上的资源有着同等的使用权。一种高效的运行方式是通过粗粒度的时间分片(TimeSlicing)使这些用户和程序能共享计算机资源，而不是由一个程序从头运行到尾，然后再启动下一个程序。  
   便利性。通常来说，在计算多个任务时，应该编写多个程序，每个程序执行一个任务并在必要时相互通信，这比只编写一个程序来计算所有任务更容易实现

# 并发一般涉及如下几个方面：

## 多线程编程（已过时，不介绍）

## 异步编程

## 并行编程

## 响应式编程

## 数据流编程

并发的实现俩中模式 并发角度来看 基于事件触发vs基于线程的

# Atitit 三种并发编程模型 并行工作者 线程模型 流水线（Assembly Line） 事件驱动模型 函数式并发模型

1. 并发系统可以使用不同的并发模型去实现。 1

2. 并行工作者 并行工作者模型。进来的任务分配给不同的工作者 银行模式 2

2.1. 并行工作者的优势 并行工作者并发模型的优势在于理解起来比较简单。为了增加应用的并行计算，你只是需要添加更多的工作者就可以了。 2

2.2. 并行工作者的劣势 3

2.3. 委派器将任务分配给Worker，Worker单独完成任务，java 7 中 java.util.concurrent　包中好多工具都是基于此模型实现的。明显存在的存在的问题是工作器之间需要共享资源。 5

3. 流水线（Assembly Line） 事件驱动系统 医院模式 5

3.1. 流水线的优势 7

3.1.1. 1：Worker之间不需要状态共享 7

3.2. 2：有一定的顺序，有利于模型的回复 7

3.2.1. 3：符合计算机硬件的特点 7

3.3. 流水线的劣势 。回调地狱 8

3.4. Nodejs采用的模型 9

3.5. Actor 和Channel 9

4. 功能并行性（Functional Parallelism）三：函数式并发模型 9

4.1.1. 使用Java 7，我们得到java.util.concurrent包包含ForkAndJoinPool，这个可以帮助你实现跟功能并行性类似的事情。使用Java 8，我们得到一个并行streams， 10

5. 哪一个并发模型是最好的 具体取决于业务场景 11

5.1. 总结：那种编程模型好？ 11

# 任务并行库(TPL)

# 定时器（java是线程实现的）

# 线程

## 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

# 线程池

# 异步

## 异步

## 同步与锁关键字

## 5.2

# 并发容器与并发集合

## Future模式

## 

# 其他

## 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 原子变量与非阻塞同步机制

## 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

## 死锁的避免与诊断

## 

# 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，该用什么来编写并发程序、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

# 22. Java c# .net c++的并发技术 7

## 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

## 22.2. Java并发技术 8

## 22.3. 《C#多线程编程实战( (（美...【 8

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

## 22.5. 《C#并发编程经典实例》 9

# 参考资料

## atitit 并发编程原理与概论 attilax总结.docx

1.1.1.atitit.并发编程／并行计算／多核编程.doc

1.1.1.atitit.并发编程／并行计算／多核编程.doc.txt

atitit java 并发编程最近实践attilax总结.docx

atitit java 并发编程最近实践attilax总结.docx.txt

Atitit 事务管理技术主要包括数据库的恢复技术和并发控制技术.docx

Atitit 事务管理技术主要包括数据库的恢复技术和并发控制技术.docx.txt

Atitit 并发api的方向.docx

Atitit 并发api的方向.docx.txt

atitit 并发编程原理与概论 attilax总结.docx

atitit 并发编程原理与概论 attilax总结.docx.txt

Atitit 融和c# .net与java的异步与并发处理解决方案 多线程.docx

Atitit 融和c# .net与java的异步与并发处理解决方案 多线程.docx.txt

atitit 高并发之道 attilax著.docx

atitit 高并发之道 attilax著.docx.7EBBE32C9BBEAF66943529FD26635889.20180207143305297.wps

atitit 高并发之道 attilax著.docx.lnk

Atitit.jmeter 并发测试使用-----随机访问页面-随机次数.doc

Atitit.jmeter 并发测试使用-----随机访问页面-随机次数.doc.txt

atitit.单台服务器并发数据总结.wps

Atitit.并发单元测试总结.wps

Atitit.并发测试解决方案(2) -----获取随机数据库记录.doc

Atitit.并发测试解决方案(2) -----获取随机数据库记录.doc.txt

Atitit.并发测试解决方案o8t20.doc

Atitit.并发测试解决方案o8t20.doc.txt

Atitit.并发测试解决方案ｏ８ｔ (2).wps

Atitit.并发测试解决方案ｏ８ｔ - 副本 (2).doc

Atitit.并发测试解决方案ｏ８ｔ - 副本 (2).doc.txt

Atitit.并发测试解决方案ｏ８ｔ - 副本.doc

Atitit.并发测试解决方案ｏ８ｔ - 副本.doc.txt

Atitit.并发测试解决方案ｏ８ｔ.doc

Atitit.并发测试解决方案ｏ８ｔ.doc.txt

Atitit.并发测试解决方案ｏ８ｔ.wps

Atitit.并发编程的调试.docx

Atitit.并发编程的调试.docx.txt

Atitit.自动化并行 并发的三大方法.doc

Atitit.自动化并行 并发的三大方法.doc.txt

Atitit.跨语言并发模型的api 兼容性与增强草案.docx

Atitit.跨语言并发模型的api 兼容性与增强草案.docx.txt

concur.txt

Google Guava官方教程（中文版） 并发编程网 - ifeve.com.htm

Java那些不为人知的特殊方法 并发编程网 - ifeve.com.htm

paip.mysql 大并发建立连接错误的解决.txt

paip.mysql 大并发调整

paip.千万级并发网站架构.txt

paip.单台服务器多少并发.txt

paip.并发编程 多核编程 多线程 JAVA 数据结构 的 对比 以及 使用.txt

PAIP.并发编程 多核编程ExecutorService的判断线程结束.txt

paip.提升性能---- 网站并发数的总结.txt

paip.提升性能--提升并发--更换web服务器效果大好.txt

《C++并发编程实战》([美]...)【简介\_书评\_在线阅读】 - 当当图书.html

《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）》(Brian...)【简介\_书评\_在线阅读】 - 当当图书.html

为什么java.util.concurrent 包里没有并发的ArrayList实现？ \_ 并发编程网 – ifeve.com.html

从并发角度来看：JavaScript是基于事件触发的，而Java是基于线程的，JavaSc.html

大型网站每秒并发.txt

服务器并发数以及压力测试闹区 ? LUA与PHP在WEB应用的性能对比\_files

服务器长连接大并发erlang.txt

高并发下下单库存超卖现象.txt