小议 Tiger 类库的多线程支持

撰文 / 莫映

引介

就在笔者开始撰写本文的前几日，令众 Java Fans 们翘首良久的 RC 版 Tiger 终于出笼了，这也意味着 Tiger 距离真正的 Final Release 已为时不远，相信届时的“Java World”一定是一派欢腾喜庆的景象。不过在万众瞩目的那一刻到来之前，我们不妨先对这个 Tiger（RC）浮光掠影一番，以解心中之痒。

对于 Tiger 的诸多新鲜特性，很早以前就已开始见诸网络，个中要旨相信大家已然耳熟能详。而本文所选议题则是 Tiger 在类库方面的一大改进，这也是 Java 程序员时常面对的一大难题——多线程应用。

J2SE 5 类库中对并发程序的支持，依据官方网站的介绍[1]，被称作“Concurrency Utilities”。此一部分主要引入了一个名为 java.util.concurrent 的包，它来自于 JCP 的 JSR166。说到这里不得不提一下 JSR166 的掌舵人 Doug Lea，正是他那个广泛流传于 Java 圈子里的多线程库 util.concurrent，以及那本颇具影响力的《Concurrent Programming in Java》，使得 Java 在多线程应用领域最终走上了标准化道路，并成为 Tiger 的一大亮点。由于多线程历来有着广泛应用，也有着广泛的使用群体，因此毫不夸张地说，新的 J2SE 中对并发机制的支持，也许是目前为止除去语言层面（比如泛型）的最大改进。

下面就让我们对 Concurrency Utilities 的主要部分逐一考察一番。同时，我们还将对比.NET Framework。由于.NET Framework 2.0 对于多线程，在大的框架方面的改进不是很大，因此这里所提及的内容大多也可以在 1.1 版本中找到[2]。

Collection 部分

java.util.concurrent 包对原来的 Collection Framework 做了不少补充。主要体现在以下几个方面：

① *新增 Queue 接口*。该接口增加了一组额外的插入、提取、访问操作：offer，poll，peek。与以往容器操作有所不同的是，它们在遇到错误时不会抛出异常，而代之以返回表征错误的结果值。

② *新增 AbstractQueue 和两个直接派生于它的 non-blocking Queue 具现类*。抽象类 AbstractQueue 实现自 Queue 接口，内含了构建 Queue 的一些缺省操作。自它派生，你只需做一些简单的覆盖工作即可。因为是 non-blocking，也就意味着对容器的并发访问无需同步，

[1] 见 New Features and Enhancements J2SE 5：http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html
[2] 详情见 API Changes between versions of the .NET Framework：
http://www.gotdotnet.com/team/upgrade/apiChanges.aspx

因此是高效率的，同时也是线程安全的。它们分别是：ConcurrentLinkedQueue 和 PriorityQueue。前者是一个基于链表结构的 Queue；后者则维护了一个优先级队列，元素依自然序排列，或是由传入构造器的 java.util.Comparator 指定。

③ *新增 BlockingQueue 接口和五个实现了该接口并派生自 AbstractQueue 的具现类。*
BlockingQueue 对比于前述的 non-blocking，定义了对容器的同步操作。五个具现类依次是：ArrayBlockingQueue， LinkedBlockingQueue， PriorityBlockingQueue， DelayQueue， SynchronousQueue。ArrayBlockingQueue 和 LinkedBlockingQueue 行为类似，唯一不同的仅是后端存储机制的差别：一个基于有界数组，一个则基于链表。PriorityBlockingQueue 是一个不限容量的依元素逻辑顺序排列的 Queue，其实现基于 priority heap。DelayQueue 有些特殊，容器内元素必须实现 Delayed 接口，该接口仅提供了一个 getDelay 方法以供容器使用。DelayQueue 中的元素只有在超过指定时延之后才可以被获取，若多个元素均超过时延，则先访问时延最长者。SynchronousQueue 是一个简单的实现了同步机制的 Queue，用于线程间简单数据的同步传输，比如典型的 consumer-producer 模式。

④ *新增 CopyOnWriteArrayList 和 CopyOnWriteArraySet*。此二类实现了 copy-on-write pattern[3]，应用于需要对容器实现大量读取操作，同时又鲜有写入操作的场合[4]。它们无需额外的 synchronized 支持，因此效率很高。事实上，写入操作首先被施于一个后备数组，这保证了对容器的读取操作不至于抛出 ConcurrentModificationException 异常，写入改动将会在对元素的后续访问中显现出来。

⑤ *新增 ConcurrentMap 接口和 ConcurrentHashMap 具现类*。ConcurrentMap 扩展了先前的 Map 接口，并增加了一组线程安全的原子级操作：putIfAbsent，remove，replace。而 ConcurrentHashMap 则实现了 ConcurrentMap。同样，它也是一个允许并发读写的线程安全的高效率容器。并且你还可以通过一个可选的 concurrencyLevel 属性，指定对容器的更新操作的并发执行数。因而在某些场合下，你完全可以用它来取代原来的 Hashtable。

纵观 Concurrency Utilities 对 Collection 部分的扩展，着实下了功夫，无论从易用性、多样性，以及性能方面，都提供了相当不错的支持。而.NET Framework 在这方面稍显不足。除了容器种类的丰富程度稍逊一筹外，在线程安全方面也有欠缺。.NET Framework 的容器当中仅有 Hashtable 是线程安全的。而对于构建线程安全的容器，Visual Studio 2005 Beta Documentation 对开发者的建议是：对原有容器作一个 Synchronized Method 的包装，或者采用加锁的手段。而由于 Array 没有提供 Synchronized Method，因而只能通过后一种方法实现线程安全。另外，对于容器的遍历，尤其当遍历期间有其他线程修改容器时，文档的建议是：要么锁住整个遍历过程，要么细心处理异常。不管怎样，这些都是手工活儿。

Thread pool 及 Scheduling service

简单的多线程应用，通常采用创建 Thread 类实例的方法。但是，线程的创建毕竟是一项很耗资源的工作，当需要面对大量有待执行的异步任务时，比如服务器程序，这种原始行为很容易因为资源耗尽而导致服务器性能不彰，甚或抛出 OutOfMemoryException 异常而使程序

[3] 见《Concurrent Programming in Java》第 2 章，2.4.4 节
[4] 一个典型的应用场景是监听者模式

崩溃。为此，java.util.concurrent 包提供了一个灵活的任务分发框架，包含了对 Thread pool 和 Scheduling service 的支持：

① *几个基本的接口*。首先是 Executor，它提供了最基本的 execute 方法，该方法传入 Runnable 类型的参数。接口 ExecutorService 扩展了 Execute，并加入了若干生命周期管理特性，比如 shutdown 操作，从而为构筑真正的服务应用提供了便利。另外两个基本接口是 Callable 和 Futrue。前者类似于 Runnable，不同的是它允许返回结果或是抛出异常。后者类似于一个异步任务的句柄，提供了控制任务的各种方法，包括：等待任务执行完毕，查询任务情况，获取执行结果，终止任务执行等等。

② *几个辅助类*。FutureTask 是一个能将 Callable 转换成 Future 的 adaptor，它实现了 Future 和 Runnable。AbstractExecutorService 则实现了 ExecutorService，并利用 FutureTask，提供了若干缺省实现。Executors 提供了产生各式 Executor 的工厂方法，这些 Executor 实现了多种任务执行方式：一个线程处理一个任务（即 thread-per-task），单一线程处理所有任务，按指定时延逐个处理任务，以及各种类型的线程池。它们事实上都是 ThreadPoolExecutor 类（或是其子类 ScheduledThreadPoolExecutor）的实例，只是配置参数有所不同。ThreadPoolExecutor 实现了 ExecutorService 接口，并且派生自 AbstractExecutorService，Executors 还提供了其他辅助函数，比如：将不同类型的实例转换成 Callable 类型。

Concurrency Utilities 中的 Executor Framework 简化了异步任务的管理，并实现了任务执行与任务提交、管理的解耦。这使开发者得以专注于具体任务的执行逻辑，而无需关心底层管理机制。.NET Framework 也提供了类似的线程池机制，它主要依靠的是 System.Threading.中的 ThreadPool 类所提供的 QueueUserWorkItem 方法，但是从灵活性、扩展性和功用方面来看，还稍显单薄了一些。

常用同步设施

Concurrency Utilities 提供了很多常用的同步设施，比如：用于限定访问特定资源的线程数目的 Semaphore；用于同步一组共同执行的工作线程的 CyclicBarrier；用于在两个线程间同步交换简单数据的 Exchanger；以及可以实现多种线程同步用途的 CountDownLatch。另外，还有一个 java.util.concurrent.locks 包专门提供了功能丰富的 Lock 机制。

这一方面，.NET Framework 从一开始就已提供了许多同步设施，尽管实现机制有所不同，但功能类似，比如：Monitor，Mutex，Semaphore，ReadWriteLock。只是在灵活性和功能上较之对手还有所欠缺。另外，在.NET Framework 中一个比较有特色的工具类是 Timer，与 TimerCallback delegate 相结合，它实现了定时执行指定任务的机制。

Atomic 部分

Concurrency Utilities 中还专门提供了一个 java.util.concurrenct.atomic 包，包含一组以线程安全的方式存取单个变量的原子级操作，它们本质上是 volatile 概念的延伸。比如：AtomicBoolean，AtomicInteger，AtomicLong，AtomicReference。同时，它还将 atomic 扩展到了 Array，比如：AtomicIntegerArray，AtomicLongArray，AtomicReferenceArray。.NET

Framework 也提供了类似机制，比如：Interlocked。

Concurrency Utilities 对原有 Java 的多线程机制还做了很多其他方面的改进，比如：对传统 Thread 类的改进，对 JVM 的改进，以及提供了作为旧有 GroupThread 捕获默认异常方式的替代实现的 UncaughtExceptionHandler，凡此种种，不一而足。

结论

撰写并发程序具有固有的复杂性。在 Tiger 之前，Java 对并发应用的支持仅仅停留在较低层面，许多基础设施都需要自己手工完成，从轮子造起。J2SE 5 的出现改变了这种窘境，它并未从语言层面做文章，而是通过提供扩展类库的方式，为构建更易使用、更易扩展、更高新能的并发程序提供了较高层面的相当不错的支持。各式工具一应俱全，这令 Java 程序员告别了多线程程序设计的刀耕火种的时代。另一方面，.NET Framework 则秉承了微软“Do more with less”的一贯风格，已简单实用为原则。因而，尽管从灵活性、扩展性、多样性等方面来看，其对并发应用的支持目前还逊于 Java，但简洁性和易用性突出，对于一般应用而言，也足以应对。不过以笔者愚见，如果.NET Framework 今后能在容器的线程安全方面，线程池的功用方面，以及同步设施的多样性方面有所提高，那将会是一件锦上添花的好事情。

参考文献

A First Look at JSR 166: Concurrency Utilities：
http://today.java.net/pub/a/today/2004/03/01/jsr166.html
Taming Tiger: Concurrent collections：
http://www-106.ibm.com/developerworks/java/library/j-tiger06164.html
Taming Tiger: Default exception handling in threads：
http://www-900.ibm.com/developerWorks/cn/java/j-tiger08104/index_eng.shtml
Put JDK 1.5's Executor to Work for You
http://www.fawcette.com/javapro/2004_08/online/bgoetz_08_04_04/default_pf.aspx
JSR 166: Concurrency Utilities：
http://jcp.org/en/jsr/detail?id=166
New Features and Enhancements J2SE 5
http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html
.NET Framework Class Library Reference
http://lab.msdn.microsoft.com/library
Doug Lea, Concurrent Programming in Java, Addison-Wesley, 1999.11