创建高性能的 J2EE 应用系统

Research on Building J2EE Application with High Performance

李晨阳　焦海星　（长沙 湖南大学软件学院 410082）

摘要：对于 J2EE 应用系统来说，性能问题是必须考虑的一个重要问题。本文将从 J2EE 应用程序的设计和部署角度出发，探究产生这些性能问题的根源，并提出一些原则来提高 J2EE 应用的性能。

关键词：性能　J2EE

1　概述

J2EE 是一个基于组件的体系结构，通过创建和组织 J2EE 组件来创建 J2EE 应用程序。它为搭建具有可伸缩性、灵活性、易维护性的应用系统提供了良好的机制。J2EE 在允许人们建立多层的面向对象的系统方面是非常成功的，但它在隐藏复杂性的同时也导致了开发人员更易于陷入严重的性能陷阱。对于 J2EE 应用来说，性能是必须考虑的一个重要问题。

对于 J2EE 应用的性能优化来说，其目标主要是提高下面几个指标：并发用户数量、吞吐量、可靠性。也就是说，让应用系统更快地为更多的用户提供服务，且保证服务过程不会中断。

2　高性能应用的设计原则

当前 Sun 颁布的最新 J2EE 规范是 JavaTM 2 Platform Enterprise Edition，v 1.4，在应用的设计上我们将遵从 J2EE 规范，从 J2EE 各个组件的性能角度出发，探讨 J2EE 应用系统的优化设计原则。

2.1　Servlet 和 JSP

2.1.1　Servlet

为了在编写 servlet 时得到良好的性能，首先要避免使用 SingleThreadModel 接口。SingleThreadModel 接口是一个标记接口，用来告诉 Web 容器在 Service( ) 方法里面的代码不是线程安全的。在使用 SingleThreadModel 时，因为容器并不是安全进程，容器将创建和缓冲大量的 Servlet 类实例。一个 Servlet 类的多个实例可能会同时访问相同的基本类或相同的数据，因此管理 Servlet 的多个实例会增加 Web 容器的负担。另外，在 Servlet 类中应尽可能避免应用 getRomoterHost( ) 方法。因为它要执行一个远程的 DNS 来匹配 IP 地址，这个操作一般要花费几秒钟的时间，这会给客户端带来不必要的延迟。

2.1.2　JSP

在应用中，JSP 所提供的功能是以 Servlet 为基础的。JSP 首先编译成 Servlet，编译时增加了少量的代码。设计 JSP 时应遵循以下原则：

（1）在 HTML 页面中不要过多使用 Java 代码：将所有的 Java 代码直接放在 JSP 页面中，是 J2EE 规范所允许的。但是这样将会导致代码难于阅读，难于理解。在 HTML 页面中减少 Java 代码的方法是编写独立的 Java 类来实现逻辑运算。

（2）选择合适的 include 机制：最好将页眉、页脚和导航条内容存储在单个文件中，并且不要重新动态产生它们。一旦将这些内容存储在各个独立的文件中，使用下面 include 机制中的任何一个就能在所有的页面中引入它们：

include 指令：<% @ include file = " filename" %>

include 行为：<jsp：include page = " page. jsp" flush = " true" />

当 JSP 正在转换成 Servlet 时，第一种 include 机制包含指定文件的内容，对于第二种 include 机制来说，当该页面执行后时，页面包含了用 Response 产生的内

容。当被包含的页面不太改变的时候,推荐使用第一种 include 指令方式,这种方式比较快,性能较好;当被包含的文件经常改变时,并且在执行主页但不能确定所要引入的页面的时候,使用第二种 include 行为方式。

(3) 合理使用自定义标签库:JSP 自定义的标签库可以封装大量的、复杂的 Java 操作在一个 Form 里面,可以有效地实现 Java 程序员和 Web 设计人员工作的划分。然而,在页面上应用的每一个标签,Web 容器都必须创建一个新的标签句柄对象或从标签缓冲中提取它。因此,过多的应用自定义的标签将会带来不必要的资源浪费。

2.2 实体 Bean

实体 Bean 用于数据存储和处理事务。当访问实体 Bean 时,会增加网络流量和其他开销,从而造成系统性能下降的可能。我们可以采用以下方法来实现对实体 Bean 的优化设计。

2.2.1 使用容器管理的持久性 CMP (Container – Managed Persistence)

使用 CMP 不仅可以减少代码的编写量,而且可以允许容器进行很多优化,包括数据库访问代码的优化。容器可以存取实体 Bean 的内存缓冲区,监控发生的变化,在提交事务之前再把缓冲区写到数据库中,这样就可以减少开销很大的数据库访问操作。另外,使用 CMP 还可以优化查找(Finds)方法。查找一个实体 Bean 通常涉及两个与数据库有关的操作:在数据库中找到相应的记录,把主键取出来;把记录数据取到缓冲区。使用 CMP,就可以在一次数据库存取中完成这两步操作,把主键和记录数据全部取出。

2.2.2 遵循粗粒度原则

由于调用 EJB 的方法都属于远程调用,对于大多数细粒度(fine – grained)的对象交互来说调用中间件的开销是巨大的。为了避免这类问题,在设计 EJB 时,每个 EJB 应代表一个独立的业务对象,有独立的特性和生命周期。具有依赖性的对象不能用 EJB 表示,它应该作为一个 EJB 内部的 Java 类来实现。客户端除了对 EJB 进行远程调用,还要进行一些检查操作,比如访问控制、事务处理以及激活/休眠等。因此,在设计中要遵循粗粒度设计原则,在单个方法调用过程中尽量增加数据传输的数量,把多个方法合并以减少方法的个数,减少方法的调用次数来提高性能。

2.2.3 使用会话 Bean

通过网络直接访问实体 Bean 不利于系统的性能,当客户端远程访问实体 Bean 时,每一个 get 方法都是远程调用。通过用会话 Bean 封装对实体 Bean 的访问能够减少系统的远程调用,会话 Bean 替远程客户端执行创建、读取、更新、删除等操作,减轻了网络的负担,提高了系统的性能。

在实际应用中,如果需要从数据库中调用多行数据,我们可以利用会话 bean 把所有需要访问的数据打包成单个调用,以此作为一个实体 Bean 本地对象的 finder 方法的返回结果,以解决通过实体 bean 进行远程调用需要消耗的大量资源的现象。

此外,还可以利用会话 Bean 来处理大量的数据操作。例如:用一个实体 Bean 来表示一份采购订单,假设当天有 100 份订单,当用户需要得到当天所有订单的货物名称时,就得检索 100 个实体 Bean,这是非常低效的。此时如果使用会话 Bean,那么就可以通过 JDBC 一次得到当天所有订单的货物名称。这样不但可以减少数据库访问次数,而且实现了业务逻辑的封装。这样就能提高系统的效率,增强系统的稳定性,使系统更易维护和使用。

2.3 设计模式

J2EE 应用系统中包含很多的组件,这些组件通过网络存取和修改数据。这往往导致对 EJB 的远程调用,这样的远程调用影响了整个应用的性能,远程调用数量的增加也会增加网络的流量。我们可以利用设计模式来减少访问 J2EE 应用程序的开销,提高性能。

(1) 快车道读取(Fast – Lane Reader)模式:通过 DAO 直接访问那些更新不是很频繁的数据,从而避免了使用 EJB 所带来的负担,加快数据的访问。

(2) 一页一页迭代显示(Page – by – Page Iterator)模式:要访问一个远程大数据列表,每次只检索其一页子列表,将列表属性构建成值对象以达到有效访问的目的。

(3) 值对象(Value Object)模式:通过将 EJB 的部

分数据封装在一个本地访问的实例(值对象)中,在访问数据的时候不对 EJB 进行访问操作,而是从值对象中获取数据,从而减少了网络传输的负担。

3 高性能应用的编码原则

J2EE 应用是使用 Java 语言编写的涉及到数据库操作的程序。因此,J2EE 应用遵循同传统 Java 和数据库设计相同的性能原则。良好的编码技术和数据库的效率对于编写高性能 J2EE 应用来说是先决条件。

3.1 Java 编码原则

Java 是使用紧凑字节码的解释性语言。速度通常是 Java 应用的最大问题,编码技术影响了 Java 应用的性能。下面是开发 J2EE 应用时需要注意的基本编码原则:

(1) 避免创建不必要的对象:创建对象是一个很耗费资源的操作。尽管 JVM 在创建对象的方面效率很高,但是,大量对象的创建仍然会占用系统的资源。因此,应该尽可能的重复利用对象,减少不必要对象的创建。

(2) 避免应用同步:系统要实现同步就必须序列化所有当前正在执行的线程,这将会降低系统的可伸缩性,并且设置同步也需要耗费 JVM 的大量的资源,因此应该尽量避免应用同步。

(3) 对象使用完之后,应该释放指向对象的全部引用。不能释放对象会使垃圾回收器不能回收这些对象。使用完引用之后,把引用设为空,会使垃圾回收器的效率更高。

3.2 数据库编码原则

应用中的数据库可能包含上千个表和上百万行数据,因此在应用中数据库的设计会对 J2EE 应用系统性能有很大影响。构建数据库时应该注意以下的几条原则:

(1) 在返回数据记录时,应该避免同时返回大量的数据记录。因为返回大量的数据将使服务器花费大量的资源和时间把它们保存在内存中,降低了系统的性能。如果应用确实需要返回大量的数据的话,应该考虑首先返回主关键字,然后根据需要提取单个的行。这种解决方法并不会减少系统整个的工作量,相反它把整个的工作量化整为零,大大的减少了反应时间。

(2) 由于访问数据库是应用服务器执行的操作中开销最大的操作,因此对数据库的存取操作应尽可能的最小化。如果在应用需要引用一些经常要用到的并且在很长时间内不会有变化的数据,就应该缓存这些数据,避免每次用到时都要进行读取操作。

4 高性能应用的部署原则

应用程序的设计和编码技术决定了系统的性能,但是仅有它们是不够的,应用程序在应用服务器上的部署也会影响系统的性能。应用服务器都提供了丰富的调试功能,通过调整功能参数,能够使服务器更适合运行环境以及应用程序。我们以 WebLogic 应用服务器为例来部署高性能的 J2EE 应用。

4.1 WebLogic Server 调优

(1) 修改执行线程数。由于应用服务器能够同时为多个并发的用户请求提供服务,而创建线程是一个很消耗资源的操作,所以应用服务器提供了可执行线程数调节,用来控制并发用户的各种请求。线程数的大小限制了应用服务器的工作量,所以必须找出一个最佳的平衡点。如果超越这个平衡点,则线程的上下文切换将产生大量开销,从而影响性能。提升服务器线程数可以增加应用程序的性能,但是在提升数量前有许多因素要考虑。在服务器上可以并行处理的线程数取决于服务器硬件的 CPU 性能,有效的处理器越多,可以给服务器的线程越多,性能的提高就越大。

(2) JDBC 连接池设置。由于 J2EE 应用支持多个并发用户,并且每一个应用都有不同的数据库访问要求,因此数据库连接池的规模和连接池中连接的数量会极大地影响应用的性能。在优化应用性能的实践中,JDBC 连接池的规模往往是对应用的整体性能表现影响最大的因素之一。

正确配置 JDBC 连接池可以增加 WebLogic Server 应用程序性能。连接池大小的设置应该足以容纳所有线程对连接的要求。如果所有对数据库的访问能够在缺省的执行队列中得以实现,则连接数应为执行队列中的线程数。为了避免在运行期间对连接进行创建和删除,可在初始时即将连接池设置为其最大容量。如

果可能的话,应确保参数 TestConnectionsOnReserve 被设置为假(false,这是缺省设置)。如果此参数设置为真(true),则在连接被分配给调用者之前,都要经过测试,这会额外要求与数据库的反复连接。

(3) EJB 调优设置。通过对 EJB 的调优,可以提高系统的性能。EJB 调优包含两个方面的内容:

① EJB 池的设置:由于无状态会话 Bean 和消息驱动 Bean 在提出请求时需要系统提供适当类型的 Bean 实例,因此,为了提供流畅的服务,负责提供 Bean 实例的 Bean 缓冲池必须足够大,否则的话,有可能会出现许多业务等待排队的现象。对于无状态会话 Bean 和消息驱动的 Bean 来说,两个很重要的性能选项就是缓冲池的大小和预先装入缓冲池的 Bean 的数量。这些池的大小可以在部署描述符 weblogic – ejb – jar. xml 的 max – beans – in – free – pool 属性下找到。可以把这些池的原始大小设置得很高,在服务器启动时,就创建一定数目的池,虽然增加了内存的使用但潜在提高了性能。

② 缓冲区的设置:由于实体 Bean 和有状态会话 Bean 都要维持一些必要的状态信息,所以应用服务器必须将它们保存在内存缓存中或临时将它们存储到磁盘中,这个缓存可在 weblogic – ejb – jar. xml 文件的 max – beans – in – cache 属性下设置。如果缓存设置的太低,在数据读取操作过程中,缓冲区内的钝化和激活操作就会过于频繁,对系统的性能造成很大的影响。

4.2 垃圾回收策略的选择

随着各种数据不断地被装入内存,应用服务器的堆栈很快就会被填满——这时候就要垃圾回收器来释放不再使用的资源。Sun JDK 1.4 定义了两种垃圾回收策略:minor 和 major。

minor 垃圾回收策略通过一个名为 copying 的过程执行,效率较高;而 major 垃圾回收策略则通过一个名为 mark compact 的过程执行,对虚拟机的工作压力较大。堆栈根据对象的存储时间分两个区:第一个是 young generation,这里的对象创建之后很快就会被删除;第二个是 old generation,在这里对象被保存到比较稳固的内存区域。young generation 通过 copying 执行 minor 垃圾回收,而 old generation 通过 mark compact 来执行 major 垃圾回收。因此,在调整垃圾回收机制时,我们的目标应该是调整两个分区的大小,尽可能多用 minor 垃圾回收策略,少用 major 垃圾回收策略。

4.3 利用服务器集群提高性能

如果应用程序有大量的用户,他们的性能请求使单个服务器使用达到最高峰,这时就需要使用集群。WebLogic 具有让应用程序在多个服务器上运行并且能够跨越不同服务器分配负载的功能。集群还提供额外的故障排除功能,当一个服务器瘫痪时,就有其他的服务器来填补它的空白。

大多数应用服务器都提供了群集的特性,并且可以在多个层次上采用群集。服务器群集采用各种不同的算法来实现负载均衡,例如 round robin、random、sticky bit、server load 等。

5 结论

本文对 J2EE 应用系统的性能问题进行了研究,分析了在开发和部署过程中困扰系统性能的原因,提出了解决的方法。在软件开发和部署过程中合理有效地利用这些方法,对于软件设计与开发者来说,可以提高系统的性能,改善系统的灵活性,使得系统能够很好地适应各种解决方案;对于应用程序的部署者来说,由于性能优化减少了系统对容器和底层系统的负载的要求,使得应用程序的部署者可以选择更加合适的基础设施,节约了资金。

参考文献

1 [美]Joseph J. Bambara 著,刘堃等译, J2EE 技术内幕[M],机械工业出版社,2002.6。

2 飞思科技产品研发中心,EJB 应用开发详解[M],电子工业出版社,2002.1。

3 Joe ZuffoLetto,BEA WebLogic Server Bible[M],电子工业出版社,2003.1。