基于池技术的高效 N 层 Web 应用体系研究

谢金晶，张艺瀕
（武汉大学 计算机学院，湖北 武汉 430072）

摘 要：对线程、内存、数据连接等宝贵资源的低效使用已成为 B/S 体系结构应用软件的主要性能瓶颈。而池技术正是解决这一问题的有效途径。介绍了基于池技术扩展而来的线程池、实例池、连接池的基本原理，阐述了如何将其运用于 N 层构架体系中相应层，优化整体的系统性能，并对存在的问题提出了改进算法。最后针对现有的池中突发的资源管理调度方法的不足，提出了基于最高效益的调度算法。

关键词：线程池；实例池；连接池；调度算法
中图分类号：TP302.1 **文献标识码**：A **文章编号**：1673－629X(2007)01－0133－03

Research for High Efficient *N* Layers Web Application System Based on Pool Technology

XIE Jin-jing, ZHANG Yi-bin
(College of Computer Science, Wuhan University, Wuhan 430072, China)

Abstract: The main bottleneck of application software, which is based on the B/S architecture, is that the ineffective use of the precious resources such as thread, memory, and data connection. Moreover, the pool technology is an efficient method to conquer this pitfall. Firstly this paper introduces the theory of thread pool, instance pool, connection pool. How to use the pool technology on the corresponding layer of the *N* layers truss system to optimize the performance of the whole system is also discussed. At last, put forward a new resource scheduling algorithm based on benefit optimization aiming at the shortage of the existing pool resource scheduling algorithm.

Key words: thread pool; instance pool; connection pool; scheduling algorithm

0 引 言

随着互联网技术的高速发展，各类商务网站的访问量也日益增多，这种批量、并发性的访问使得商务网站对用户的响应速度会明显变慢，甚至有时可能会引起系统的崩溃。其主要原因就是对如线程、内存、数据连接等宝贵资源在空间及时间上的低效使用。对池技术的恰当使用，可以有效地减少对象生成和初始化时的消耗，提高系统的运行效率。

池最基本的思想就是预先建立一些资源放置于内存对象中以备使用，并保存用过的对象，等下一次需要这种对象的时候，再拿出来重复使用，从而一定程度减少频繁创建对象所造成的开销[1]。例如线程、EJB 实例、JDBC 连接等。若将池想象成一个保存着各种需要的对象的容器，则可通过对这些对象的复用提高系统性能。

文中提出了基于 Java 技术，及利用池技术扩展而来的线程池、实例池、连接池优化 N 层构架的 WEB 应用体系的思想，并给出了相应的改进策略。

由图 1 可以看出，这样的多层次的构架体系，其层次更为清晰，而且层与层之间的耦合也更为松散，因此也更有弹性。由此，笔者提出：

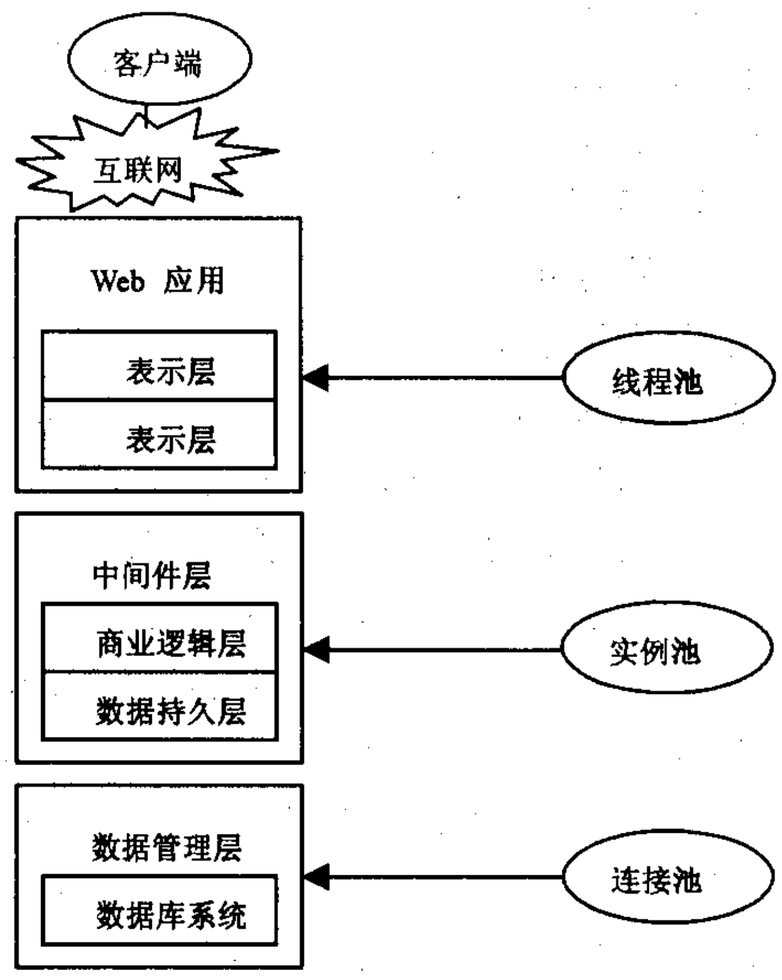

图 1 N 层构架技术体系

(1) 在与客户端直接相连的 WEB 应用层上，利用线程池技术处理客户请求。

收稿日期：2006－04－04
基金项目：湖北省自然科学基金(2005ABA238)
作者简介：谢金晶(1985－)，女，湖北随州人，研究方向为信息安全；导师：董文永，副教授，研究方向为智能计算。

(2) 在中间件层上,用 EJB 担当此角色,采用对象池的技术提高系统性能。

(3) 在数据管理层上,采用连接池技术减少对数据库的连接操作来满足系统要求。

但是在每个层次运用相应的池技术时,由于现有池技术都存在着同样的问题:短时间内面临突发大量的资源请求时无法很好地工作,因此文中在最后提出了基于最高效益的调度算法,使系统性能进一步得到改善。

1 池技术在相应层的运用

1.1 利用线程池技术处理客户请求

浏览器请求与应用服务器响应客户端请求的方式是 WEB 应用中的典型信息交换方式。此方式需应用服务器每收到来自客户端浏览器的请求后,创建一个服务线程接受来自客户端的请求,并在处理完毕后将其移除。而用户发起的请求经常具有高密度、短时间的特性,这便会导致频繁地建立与关闭线程,会极大地减低系统性能。

可以采用线程池技术来解决这些由于处理请求耗时而影响性能的问题[2~7]。

假设在一台服务器完成一项任务的时间为 T:T_1 为创建线程的时间,T_2 为在线程中执行任务的时间,T_3 为线程销毁的时间。则有 $T = T_1 + T_2 + T_3$。线程池技术正是用缩短或调整 T_1,T_3 时间来提高服务器程序性能的技术。

(1) 缩短 T_1:在应用服务器启动的时候,初始化一个线程池,并创建一些预备线程等待应用服务器选择并处理来自客户端的请求,这样大量的创建线程的时间 T_1 便被安排在了服务器启动时,节省了时间。

(2) 缩短 T_3:线程处理完毕后,应用服务器并不将其销毁,而是将这个线程归还给线程池,等待再一次被选取。这样不仅可以减少 T_3,还减少了创建线程的个数。

1.2 调节 EJB 实例池的大小来优化系统性能

EJB(Enterprise Java Bean)是一种服务器端组件的体系结构,作为一种优秀的中间件,经常在 N 层构架体系中担当着中间件的角色。

在 EJB 技术规范中依照自身特性的不同,将其分为三类:会话 EJB(Session EJB),消息 EJB(Message Driven Bean),实体 EJB(Entity Bean)。其中,会话 EJB 根据释放对应客户端的状态分为有状态会话 EJB(Stateful Session Bean)和无状态会话 EJB(Stateless Session Bean)。

对无状态会话 EJB,若用户发出请求,则 EJB 容器会在实例池中没有闲置的无状态会话 EJB 的实例的情况下才创建新的实例,否则便选择闲置的实例。因此可知无状态会话 EJB 有可被多用户共享、重用的特性。而实体 EJB 与无状态会话 EJB 具有此方面相似特性。

由此,可采用 EJB 实例池技术来解决创建和移除 EJB 实例消耗大量资源的问题[8]。而指定合适的 EJB 实例池中实例数量的值,即设置一个最优的最大值,是提高系统应用性能的关键。若此值过小,当有新客户请求到来时,由于实例池中的 EJB 实例的数量已达上限,则客户端的请求必须排队,这样会使客户端得到服务器响应的时间变长。若此值过大,在实际应用中都不可能达到此值,则反而会极大地消耗服务器端的系统资源,降低系统性能。

1.3 利用连接池技术优化数据库连接

在使用 Java 语言进行和数据库有关的应用开发中,一般都使用 JDBC 来进行和数据库的交互。由于在 WEB 应用系统中,不可避免地要频繁访问数据库,如每次接受到请求,就向数据库申请一个连接,执行完后再断开,这样将大量耗费的时间空间资源。因此需用连接池来解决此问题[9]。

(1) 建立连接池。

首先要建立一个静态的连接池。静态是指池中的连接是在系统初始化时就分配好的,且不能够随意关闭这些连接。Java 中提供了很多容器类可以方便地用来构建连接,如:Vector,Stack 等。在系统初始化时,根据配置创建连接并放置在连接池中,以后所使用的连接都是从该连接池中获取的,这样就可以减少建立、关闭连接造成的开销。

(2) 使用引用计数优化连接池管理。

基于已建立的连接池,则可以提出一套自定义的分配、释放策略。笔者在此应用引用计数(Reference counting)的设计模式来管理连接的分配和释放。具体实现思想为:将连接池分为空闲池和使用池。当连接池初始化时,所有的数据库连接均放入空闲池中,并为每一个连接建立一个引用计数来标记使用此连接的用户个数。当客户请求数据库连接时,首先看空闲池中是否有空闲连接,若有空闲连接则将其分配给客户,并将其移入使用池中,增加其引用计数。若空闲池中无连接,就在使用池中寻找一个引用计数最小的连接给客户。但此方法仍有缺陷,如当使用池中引用计数最小的连接个数不是一个,而是很多个时(此情况多出现在用户较少时),此方法未给出如何再次选取最优连接。基于此,笔者提出给每个连接再加上一个属性值 last_use_time,来记录从此连接最后一次被使用到现在的时间。则当最小引用计数数据库连接个数多于一个时,可从 last_use_time 中最大的一个使用,以提高效率。

2 池中资源的调度

线程池、EJB 实例池、连接池是由池技术扩展而来的,则其有着池的固有特性和优点。

但也同时有着共同的不足,如当突发的在短时间内有 N 个申请者请求使用资源,而此时可用资源数只有 M,$M < N$,并且 M 已达池内所能创建的最大资源个数,若再增大的话会影响到系统的整体运行效率,那么以上介绍的方法即都不能解决了。

因此,笔者针对池中突发的资源管理调度问题,提出了基于最高效益的调度算法[10~12]。

2.1 效益函数

文中提出了在性能指标,即时间响应比、空间占用率及优先级,与调度所能获取的效益之间建立函数关系的效益函数。则可通过优先调度最大效益的任务达到性能的最优值。定义函数为:

$$R_i = \alpha \times \left(\frac{W_i}{S_i}\right) + \beta \times \left(\frac{1}{B_i}\right) + r \times A_i$$

其中有:

(1) W_i 为申请任务的等待时间。当申请者提出申请任务时,Wi 开始计时,当任务获得所需资源时停止计时。

(2) S_i 为预计此任务需占用资源的时间。

(3) B_i 为预计任务在占用资源时所需的系统空间开销。

(4) A_i 为任务优先级。事先根据可能出现的各种任务的本质特性为其设定优先级。则当有申请任务到达时,便可根据其固有的特性得到其对应的优先级。

则可知 $\left(\frac{W_i}{S_i}\right)$ 描述了任务的时间响应比,则任务等待时间 W_i 越长,占用资源时间 S_i 越短,其时间响应比就越大,因而优先调度所获效益就越大。且任务所需系统空间开销越大,则可获得的效益越小,应推迟调度,故用 $\left(\frac{1}{B_i}\right)$ 描述任务空间利用率。而 A_i 描述了任务的优先级,其值越大,级别越高,则越早调度所获利益也就越大。因此为了获取最大效益 R_i,则必须权衡时间响应比、空间利用率及任务优先级所占的比重。则有 $\alpha + \beta + \gamma = 1, 0 < \alpha, \beta, \gamma < 1$。

2.2 调度算法

本算法在效益函数的基础上还需引进 $T\_$ wait, $T\_$ use 的概念。其中,$T\_$ use 为任务所能占用资源的最长时间,当任务占用资源时间超过 $T\_$ use,还未归还时,若系统中等待序列不为空,则需强迫其归还,并将其插入等待队列中。$T\_$ wait 为任务所能等待的最长时间。当任务所等待的时间与 $T\_$ wait 相等时,则为了防止饥饿发生,必须采取一定的策略,使此任务得以调度。另外,本算法将任务的优先级设为可变项,当任务刚到达时得到一个初始优先级,但在其调度中会发生变化。

设此时有 N 个请求任务,池中有 M 个可用资源。具体算法伪码如下:

```
Task{
  for (int i = 0; i < N; i ++){
    由任务特性在系统优先级表内查找到 Ai;
    计算任务的时间响应比(Wi/Si),及空间占用率(1/Bi),计算 Ri;
  }
  TaskSort(Ri);//根据 Ri 从大到小对任务队列排序
  int front 指向队首,rear 指向队尾;
  //当任务等待序列不为空时先分配池中空闲资源
  while(front<=rear&& 池中空闲资源数 num! = 0){
    从队首中取出任务,获取资源;
    front = front + 1;
    num = num - 1;
  }
  //当任务等待序列仍不为空时分配被占用超时资源
  for(i = 0;i < M &&front<=rear; i ++){
    if(资源 i 被任务 j 站用时间 T_Uj 超过 T_use){
    T_Uj = 0;//将任务 j 使用资源的时间设为 0;
    将任务 j 取出,任务 j 优先级 Aj = Aj+1,计算其 Rj,插入等待队列;
    释放资源 i;
    从队首取出任务,占用资源 i
      front = front + 1;
    }
  }
  //flagi 为任务 i 等待超时标记;
  for(i = 0;i < N;i ++){
    if(任务 i 的等待时间 T_Wi >= T_wait){
      flagi = 1;//表明任务 i 等待超时
      Ai = Ai+1;//任务 i 优先级增加;
    }
  }
  //使等待队列中等待超时的任务得到资源
  while (front<=rear) {
    for(i = 0;i < N; i ++){
      if(flagi = = 1){
        采用 LRU(最近最少使用算法)在池中获取一个资源 j;
        将正在使用此资源的任务 k 取出,
        Ak = Ak+1,计算其 Rk,按序插入等待序列中;
        任务 i 获得此资源;
        front = front + 1;
      }
    }
  }
}
```

3 结束语

当前互联网正在由基于 B/S(Browser/Server)架构的 3 层开发模式逐渐向 N 层构架体系的应用模式发展。但如线程、内存、数据连接等宝贵资源在空间及时间上的低效使用,极大影响着客户端应用程序的性能和用户的满意度。在 Java 技术的基础上,将由池技术扩展而来的线程池、EJB 实例池、连接池技术进行优化,并运用于 N 层构架体系中相应层,可以提高整体的系统性能。

但现有的池技术仍有不足之处,如池中突发的资源管理调度问题。笔者针对此种不足提出了基于最高效益的调度算法。此算法可兼顾时间响应比、空间占用率及任务优先级三个性能指标。故可得到较短的时间响应提高用

(下转第 138 页)

系统可以定期自动删除该类别或提示用户是否需要保留该类别。对于用户收到的新垃圾邮件,系统通过建立新的类别和与其他类别合并成聚合度较高的新类别来实现动态识别内容可能发生某些变化的垃圾邮件。

3 优 点

由于系统在训练过程之前对于垃圾邮件的信息一无所知,如果用户指定某些邮件(这些邮件可能在大多数邮件过滤系统中被判定为垃圾邮件而被自动过滤)为合法邮件,那么在后续阶段,用户可以收到这类邮件而不会被本系统自动判定为垃圾邮件。同时用户可以看到所有垃圾邮件的类别,并可以指定某一或某些类别为合法邮件类别。

对于用户收到的垃圾邮件来自于一小部分垃圾邮件发送者这类情况,一个类别会自动映射到一个发送者。这样即使这些发送者发出的垃圾邮件作了一些修改,系统也可以将其归入到对应的类别。

4 不足之处

垃圾邮件的信息被存放在客户端的数据文件中,增加了存储开销。接收垃圾邮件带来的网络带宽开销是不可避免的。在训练过程中,用户需要指定以往的邮件中哪些邮件为垃圾邮件。受用户收到的垃圾邮件数量的影响,系统需要学习并积累足够多的垃圾邮件类别信息才能更有效地识别垃圾邮件。

5 小 结

本系统由于其良好的可扩展性和灵活性,实施个性化的垃圾邮件识别策略非常适合。未来的研究方向主要集中于计算邮件相似度算法的进一步改进、主要算法中可变的阈值对垃圾邮件识别的正确率和召回率的影响以及合法邮件的自动分类等。

参考文献:

[1] 中国互联网络信息中心. 第十六次中国互联网络发展状况统计报告[EB/OL]. 2005 - 07. http://www.cnnic.net.cn/index/0E/00/11/index.htm.

[2] Deepak P, John J, Parameswaran S. A Community Based Approach for Spam Filtering[C]//ICTTA 2004. IEEE International Conference[s.l.]:[s.n.],2004:611 - 612.

[3] Deepak P, Parameswaran S. Spam Filtering using Spam Mail Communities[C]//Proceedings, IEEE SAINT'05[s.l.]:[s.n.],2005:377 - 383.

[4] 刘 源,谭 强. 信息处理用现代汉语分词规范及自动分词方法[M]. 北京:清华大学出版社;南宁:广西科学技术出版社,1994:36 - 60.

[5] 林 珊,宁国宁,赵之霖. 中文分词在邮件过滤系统中的应用[J]. 华南理工大学学报:自然科学版,2004,32(增刊):113 - 116.

(上接第 135 页)

户满意度,又仅占用较少的空间获得较高的空间利用率。并且可以在不增加池中资源个数的情况下,通过对任务效益进行排序,及设定 T_wait(任务最长等待时间) 及 T_use(资源最长被占用时间),使得每个任务都可在有限时间内被响应,获得资源,从而使池的资源利用率得到了进一步的提高。

参考文献:

[1] Grand M. Patterns in Java[M]. [s.l.]:John Wiley & Sons Publish, 1999.

[2] Gamma E, Helm R, Johnson R, et al. Design Patterns Elements of Reusable Object - osiented Software[M]. [s.l.]:Addison - Wesley Publish,1995.

[3] Blumofe R D, Leiserson C E. Scheduling Multithreaded Computations by Work Stealing[J]. Journal of ACM, 1999, 46(5):720 - 748.

[4] Ling Yibei, Mullen T, Lin Xiaola. Analysis of Optimal Thr-ead Pool Size[J]. ACM SIGOPS Operating System Review, 2000, 34(2):42 - 55.

[5] 林胜利,王坤茹,孟海利. Java 优化编程[M]. 北京:电子工业出版社,2005.

[6] 高正光,李启炎. 一种多线程并发环境下的对象缓存模型[J]. 软件技术与数据库,2005(22):102 - 106.

[7] 李 昊,刘志镜. 线程池技术的研究[J]. 现代电子技术,2004(3):77 - 79.

[8] 水 超,李 慧. 对象池模式的扩展与设计[J]. 计算机工程,2004(5):26 - 28.

[9] 丁志山. JSP 数据库连接池的必要性及实现[J]. 信息技术,2005(4):112 - 114.

[10] Takefusa, Casanova A. A Study of Deadline Scheduling for Client - Server Systems on the Computational Grid[C]//Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing. LosAlamitos: IEEE, 2001:406 - 415.

[11] Subrammani V. Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests[C]//Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing. Los Alamitos: IEEE, 2002:359 - 367.

[12] 胡自林,徐 云,毛 涛. 基于效益最优的网格资源调度[J]. 计算机工程与应用,2005(7):69 - 70.