基于JDBC与设计模式的数据库连接池实现方法

欧阳宏基,葛 萌,赵 蔷
(咸阳师范学院 信息工程学院,陕西 咸阳 712000)

摘 要:目前Java数据库应用程序中存在着通过JDBC访问数据库时频繁创建、关闭Connection对象的做法。这种做法对并发访问量大的应用会带来较大的性能开销。针对这一问题,提出并实现了一个基于JDBC与设计模式相结合的数据库连接池实现方法,该方法从连接池对象和连接池管理者两个方面实现对Connection对象的管理,Connection对象使用完后不去关闭而是放回到连接池中以达到复用的目的,从而提高了Java数据库应用的效率。

关键词:JDBC;Connection;数据库连接池;设计模式;Java数据库应用
中图分类号:TP311.52 **文献标识码**:A **文章编号**:1673-629X(2011)01-0084-04

Realization Method of Database Connection Pool Based on JDBC and Design Pattern

OUYANG Hong-ji, GE Meng, ZHAO Qiang
(Information Engineering College, Xianyang Normal University, Xianyang 712000, China)

Abstract: At present, in the Java database applications exists a solution that the Connection object is frequently created and closed when the database is visited through JDBC. This solution will bring a large performance overhead for the applications which have large concurrent access volume. In view of the issue, has proposed and realized a database Connection pool method based on the combination of JDBC and design pattern. The method has achieved the management of the Connection object through the Connection pool object and the Connection pool manager, the Connection object is not closing but putting back the Connection pool after using up to achieve the purpose of reuse, thus improving the efficiency of Java database applications.

Key words: JDBC; Connection; database connection pool; design pattern; java database application

0 引 言

JDBC(Java DataBase Connectivity)是Sun公司为Java语言提供的访问各类关系数据库的标准接口[1]。JDBC的最大特点是它独立于具体的关系数据库。与ODBC(Open Database Connectivity)类似,JDBC API中定义了一些Java类分别用来表示与数据库的连接(Connection)、SQL语句(SQL Statement)、结果集(ResultSet)以及其它的数据库对象[2,3],使得Java程序能方便地与数据库交互并处理所得的结果。

目前Java数据库应用中普遍存在访问数据库时加载驱动程序、创建连接对象(Connection)、创建Statement对象、处理SQL语句、关闭连接等这些必需的操作。其中创建Connection对象的开销是最大的,因为Connection对象代表一条数据库连接,从计算机网络原理的角度理解,创建Connection在传输层采用面向连接和确认的TCP协议。尤其是在分布式网络环境中,Connection对象就仿佛在客户端和数据库服务器端搭建了一座桥,频繁的建桥和拆桥是很浪费时间的。对于简单的数据库应用,由于并发访问量小,需要时创建Connection对象使用完毕释放连接对象不会带来明显的性能开销。但对于大访问量的数据库应用,频繁的建立连接、关闭连接会造成很大的性能开销,从而影响系统的效率。

针对上述问题提出一种基于JDBC与设计模式相结合的数据库连接池方法,该方法的最大特点是实现连接对象的复用,也就是说数据库访问完成后并不真正意义上关闭连接,而是将连接对象交还给连接池。

收稿日期:2010-05-15;**修回日期**:2010-08-17
基金项目:陕西省教育专项科研计划项目(09JK811);咸阳师范学院专项科研基金项目(07XSYK283)
作者简介:欧阳宏基(1982-),男,陕西太白人,助教,研究方向为web应用、网络集成与数据库技术;赵 蔷,副教授,研究方向为软件工程、图像处理。

1 数据库连接池整体设计思想

1.1 建立连接池

建立一个静态的连接池,即池中的数据库连接是在系统初始化时创建好的,并且不能够随意关闭。

Java集合框架中的很多类都能充当这个连接池,例如 Vector、Stack、LinkedList 等[4]。采用 LinkedList 集合框架来保存 Connection 对象,因为它基于链表这种数据结构具有快速插入和删除的优点,符合取出和保存 Connection 对象的要求。以后需要数据库操作时直接从集合中取出 Connection 对象,使用完毕再放回到集合中,这样只相当于对内存进行操作,可以避免连接随意创建、关闭造成的开销。

1.2 初始化参数的设置

数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中,这些数据库连接的数量是由最小数据库连接数来设定的[5]。无论这些数据库连接是否被使用,连接池都将一直保证至少拥有这么多的连接数量。连接池的最大数据库连接数量限定了这个连接池能占有的最大连接数,当应用程序向连接池请求的连接数超过最大连接数量时,这些请求将被加入到等待队列中。数据库连接池的最小连接数和最大连接数的设置要考虑到下列几个因素[6,7]:

(1) 最小连接数是连接池一直保持的数据库连接,所以如果应用程序对数据库连接的使用量不大,将会有大量的数据库连接资源被浪费。

(2) 最大连接数是连接池能申请的最大连接数,如果数据库连接请求超过此数,后面的数据库连接请求将被加入到等待队列中,这会影响之后的数据库操作。

(3) 如果最小连接数与最大连接数相差太大,那么最先的连接请求将会获利,之后超过最小连接数量的连接请求等价于建立一个新的数据库连接。不过,这些大于最小连接数的数据库连接在使用完不会马上被释放,它将被放到连接池中等待重复使用或是空闲超时后被释放。

要确定最小连接数和最大连接数,需要考虑应用的实际情况。可根据应用的日志信息获取某一时刻有多少连接正在被使用(ActiveCount)以及有多少连接被创建了(CreatedCount),最小连接数应该小于平均 ActiveCount,最大连接数应该在 ActiveCount 和 CreatedCount 之间。

1.3 连接池管理策略

连接池需要有"专人"制定相应的侧率来管理,文中定义了连接池管理者类,对于连接的分配、回收都要通过这个管理者,管理者是唯一的。当客户请求连接时,首先管理者要看池中是否有空闲的连接(已经创建好但未被使用的连接),如果有就在空连接就分配一个给用户,并且将该连接从池中移除。如果没有空闲的连接,就要看连接池中创建的连接有没有达到最大连接数,如果没有达到就调用 DriverManager. getConnection( )方法创建一个新的连接并分配给用户,同时将连接数加 1。若连接数已达到最大连接数,则此时的请求就要进入等待状态,需要等待其他线程释放连接再获取。如果等待时间超过允许的最大等待时间则通知用户本次请求失效。当用户释放连接时,将其放入连接池中(添加到 LinkedList 中),并将当前正在使用的连接数减 1。

2 数据库连接池的实现

具体的实现包括两个部分的实现:数据库连接池(ConnectionPool)的实现和连接池管理者(ConnectionPoolsManager)的实现。

数据库连接池主要实现的功能有:

(1)获取连接给客户端。

(2)回收客户端不再使用的连接。

(3)创建新的连接。

(4)关闭本连接池中的所有连接。

连接池管理者主要用于管理所有连接池对象,主要实现的功能有:

(1)注册 JDBC 驱动。

(2)创建各连接池对象。

(3) 实现连接池名字与连接池类对象之间的映射,从中获取和返回相应的连接。

(4) 关闭各连接池的所有连接,释放所有资源。

2.1 连接池实现

```
class ConnectionPool {
private int currentConnectCount; //当前连接数
private LinkedList freeConnections = new LinkedList( ); //保存所有可用连接
  private int maxConnectCount; //此连接池允许建立的最大连接数
  private String name; //连接池名字
  private String password; //密码
  private String URL; //数据库的 JDBC URL
  private String user; //数据库用户名
public ConnectionPool(String name, String URL, String user,String password, int maxConnectCount) {
//构造方法完成属性的赋值操作}
public synchronized void freeConnection(Connection con) {
//该方法将客户端用完的连接收回,并通知等待连接的用户}
public synchronized Connection getConnection( ) {
// 该方法从连接池获得一个可用连接。如果没有空闲的连接且当前连接数小于最大连接数限制,则创建新连接。如原来登记为可用的连接不再有效,则从列表中删除,然后递归调用自己以尝试新的可用连接
Connection con = null;
  if (freeConnections. size( ) > 0) {
      //获取向量中第一个可用连接
```

```
        con = (Connection) freeConnections.removeFirst();
        try {
          if (con.isClosed())
            //如果连接以关闭递归调用自己,尝试再次获取可用连接
            con = getConnection();
} catch (SQLException e) {
            con = getConnection();}
} else if (maxConnectCount == 0 || currentConnectCount< maxConnectCount) {
//创建新的连接
con = newConnection();
  }
        if (con ! = null) {
          currentConnectCount++;
        }
        return con;
}
public synchronized Connection getConnection(long timeout) {
//该方法从连接池获取可用连接,可以指定客户程序能够等待的最长时间,与前一个 getConnection()方法重载}
   public synchronized void release() {
//该方法关闭本连接池中的所有连接}
private Connection newConnection() {
//该方法通过 DriverManager 类创建一个连接
Connection con = null;
   MyConnection realCon=null;
try {
     if (user == null) {
       con = DriverManager.getConnection(URL);
//利用代理模式对这个连接进行封装,请参考3.1节说明
       realCon=new MyConnection(con,this);
}
else {
        con = DriverManager.getConnection(URL, user, password);
       realCon=new MyConnection(con,this);
   }
  } catch (SQLException e) {
      return null;
   }
    return realCon;
    }
}
```

2.2 连接池管理者的实现

```
public class ConnectionPoolManager {
private static ConnectionPoolManager instance; //该类的唯一实例
private static int clientsCount; //请求连接的客户端数量
private Vector drivers = new Vector(); //保存各数据库JDBC驱动,Vector是线程安全的
private Hashtable pools = new Hashtable(); //保存各数据库连接池
public static synchronized ConnectionPoolManager getInstance() {
//该类采用单件模式实现,如果是第一次调用此方法则返回该类的唯一实例,参见3.2节说明
if (instance == null) {
          instance = new ConnectionPoolManager();
      }
       clientsCount++;//请求客户端数量加1
      return instance;
}
private ConnectionPoolManager() {
  //私有构造方法,防止在其他类中创建本类实例,调用init()方法完成初始化操作}
public void freeConnection(String name, Connection con) {
//该方法将连接对象返回给由名字指定的连接池}
public Connection getConnection(String name) {
//该方法获得一个可用的(空闲的)连接。如果没有可用连接,且已有连接数小于最大连接数限制,则创建并返回新连接}
public Connection getConnection(String name, long time) {
  //该方法获得一个可用连接。若没有可用连接,且已有连接数小于最大连接数限制,则创建并返回新连接。否则,在指定的时间内等待其它线程释放连接}
public synchronized void release() {
//该方法关闭所有连接,撤销驱动程序的注册}
private void createPools(Properties props) {
//该方法根据指定属性创建连接池实例}
private void init() {
//该方法读取属性配置文件完成初始化}
private void loadDrivers(Properties props) {
//该方法装载和注册所有JDBC驱动程序}
}
```

3 设计模式在连接池中的应用

3.1 利用代理模式关闭连接

设计模式在软件开发当中表现为一组通过精心设计的类和接口,为重复出现的应用场景提供一个通用的解决方案[8,9],从而提供软件的复用性和可扩展性。代理模式可以给某一个对象提供一个代理对象,并由代理对象控制对原对象的引用[10]。例如现实生活中的代理公司,在客户与当事人之间起到一个中介作用,客户想联系当事人必须通过代理。代理模式的类的演化结构如图1所示:抽象目标角色的作用是声明真实目标和代理的共同接口。代理对象角色包含对真实目标的引用,从而可以操作真实目标角色。真实目标角色定义了代理角色所代表的真实对象,是客户端真正想与之交流的对象。

代理模式可以用来将 Connection 对象放回到连接池中。当 Connection 对象使用完毕后,需要将它放回到连接池中而不是通过它的 close()方法彻底关闭,否

则达不到连接复用的效果。为了使开发人员继续使用熟悉的 close( )方法又不是彻底关闭连接对象,因此需要改写 Connection 对象的 close( )方法。Connection 对象本身就是接口,不同的数据库驱动都对它做了具体实现。如果采用继承具体数据库 Connection 的实现再覆盖 close( )方法,当更改数据库时将带来较大的工作量。面向对象中强调"组合优先继承"这个观点,所以需要自定义实现 JDBC 中 Connection 接口的类。

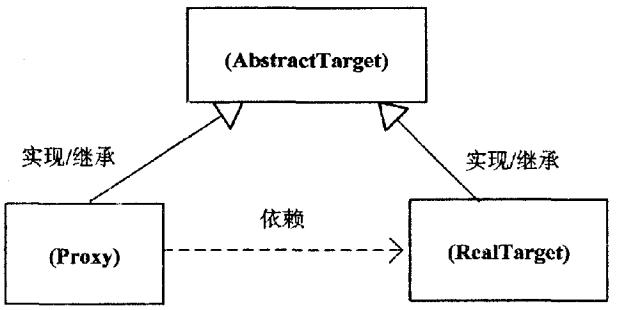

图 1　代理模式类结构图

```
public class MyConnection implements Connection {
    private Connection realConnection;
    private MyDataSource myDataSource;
    MyConnection(Connection con,MyDataSource myDataSource)
    {  this.realConnection=con;
       this.myDataSource=myDataSource; }
    public void clearWarnings( ) throws SQLException {
        this.realConnection.clearWarnings( ); }
    public void close( ) throws SQLException {
        this.myDataSource.ConnectionsPool.addLast(this); }
    ..................................
```

在 MyConnection 类中,java.sql 包中的 Connection 接口充当抽象目标角色,realConnection 是真正的连接对象(真实目标),它引用一个具体数据库的 Connection 对象。只需要重写 close( )方法,把它放入到连接池中。至于 JDBC 的 Connection 接口中的其他方法,只需要调用真正连接对象的相应方法去完成就可以了,因此 MyConnection 类是代理模式的一个具体实现。采用代理模式与直接调用真实目标对象相比在一定程度上降低了应用的性能,但它却使软件扩充变的容易。

3.2　利用单件模式管理连接池

单件模式保证一个类在软件系统的整个生命周期中仅有一个实例,并提供一个访问它的全局访问点[11,12]。这是为了解决多线程环境下,有可能引发资源访问冲突问题而提出的。单件模式的最大特点是该类的构造方法被声明为 private 访问权限,这样只能在该类内部调用构造方法和防止该类被继承而创建多个实例。并且声明一个名为 getInstance( )的静态同步方法来获取这个类的唯一实例。

Web 应用正是多线程环境,多个客户端很可能在同一时刻访问数据库,都会经连接池管理者从连接池中索取连接对象。如果存在多个连接池管理者就有可能出现连接管理混乱的现象(线程 1 通过连接池管理者 1 请求连接对象,线程 2 通过连接池管理者 2 请求连接对象,恰逢连接池中没有现成的连接对象,连接池管理者 2 很可能把管理者 1 创建给线程 1 的新的连接对象交给线程 2,导致线程 1 不能获得连接对象)。因此连接池管理者必须是唯一的,这符合单件模式的要求。

4　结束语

文中主要讨论了数据库连接池的基本原理,从连接池对象和连接池管理对象两个方面给出了实现方案并结合设计模式给出了较详细的实现代码,具备了高效、安全复用数据库连接对象的功能。但仍然有些问题没有给于解答,例如跨不同连接对象的事务问题,在此问题上可以根据实际需要采用显示的事务处理方法,每个事务独占一个数据库连接。

参考文献:

[1] 耿祥义,张跃平. Java 2 实用教程[M]. 第 3 版. 北京:清华大学出版社,2006:391-392.

[2] 谷庆华,李成贵. 基于 Java 语言实现数据库的访问[J]. 计算机技术与发展,2008,18(2):13-14.

[3] JDBC API Specification Version 4.0. Introduction to the JDBC [EB/OL]. 2010. http://java.sun.com/javase/6/docs/technotes/guides/jdbc/.

[4] 刘继华,李腊元. 一种基于 JDBC 的数据库连接池的设计与实现[J]. 计算机工程与应用,2003(7):183-1842.

[5] 罗荣,唐学兵. 基于 JDBC 的数据库连接池的设计与实现[J]. 计算机工程,2004,30(9):92-94.

[6] 习　磊,周平安. 基于 JDBC 的数据库连接池高效管理策略[J]. 计算机工程与应用,2003(30):203-204.

[7] Bloch J. Effective Java programming language guicle[M]. New York:Pearson Education Inc,2003.

[8] Gamma E,Helm R,Johson R,et al. Design patterns:Elements of reusable object-oriented software[M]. New York:Addison Wesley,1995.

[9] 曾　慰,陈维斌. 设计模式在新生报到系统中的应用与实现[J]. 计算机技术与发展,2007,17(7):178-182.

[10] Shalloway A,Trott J R. 设计模式解析[M]. 徐言声,译. 北京:人民邮电出版社,2007:241-242.

[11] 欧阳宏基,解争龙,黄素萍,等. 一种基于 DAO 设计模式与 Hibernate 框架的 Web 应用模型[J]. 微计算机应用,2009,30(3):40-41.

[12] Reilly K. Using JDBC & the Template Method Pattern for Database Access[J]. Java Developer's Journal,2005,10(3):45-46.