第 20 卷 第 3 期 五邑大学学报（自然科学版） Vol.20 No.3
2006 年 10 月 JOURNAL OF WUYI UNIVERSITY (Natural Science Edition) Oct. 2006

文章编号：1006-7302（2006）03-0062-05

基于数据库的 WEB 程序开发中连接池的设计与实现

彭利民

（广州体育学院 计算机教研室，广东 广州 510075）

摘要：在对传统数据库连接模式分析的基础上，采用一种基于数据库的连接池技术，通过在连接对象中放置预先建立的若干“ 连接 ”，并按照一定的策略有效管理这些“ 连接 ”，利用 JAVA 语言实现该方案，旨在提供一个高效的数据库连接管理策略，提高 WEB 数据库应用系统的性能.

关键词：连接池；设计模式；数据库

中图分类号：TP 311.52 文献标识码：A

Design and Realization of the Connection Pool in WEB Procedures Based on Database

PENG Li-min

(Computer Teaching and Research Dep., Guangzhou Institute of Physical Education, Guangzhou 510075, China)

Abstract: On the basis of an analysis of the traditional data-connection mode, this paper adopts a kind of connection pool technology based on database. The scheme can be realized by placing a lot of connections in connection objects beforehand, managing those connections according to some tactics effectively, and using JAVA. It can offer high-efficiency tactics for managing those connections and raise the performance of WEB database application system.

Key words: connection pool; design pattern; database

随着计算机技术的发展和WEB应用的日趋广泛，B / S结构的程序逐渐成为企业应用的主流，在访问基于数据库的WEB应用程序时，通常先建立数据库连接，然后进行相关的SQL操作数据库，最后关闭数据库连接. 如浏览动态WEB网站时，一般是通过调用WEB数据库存储的信息生成WEB页面，每个页面请求都将进行一次数据库操作. 这样连接数据库不仅要开销一定的通信和内存资源，还要完成用户验证、安全上下文配置这类任务，因而常会降低数据库服务器的性能. 同时，在程序设计中必须考虑管理每个连接，确保数据库操作的正确，如果出现程序异常而导致某些连接未能关闭，将导致数据库系统中的内存泄露，使数据库无法正常工作. 本文在对传统数据库连接分析的基础上，采用以JAVA语言为基础的数据库连接池技术，在主程序中建立数据库连接池以

收稿日期：2006-01-13

作者简介：彭利民（1976- ），男，湖南邵阳人，硕士，研究方向：数据库技术，网络与并行分布式计算.

及合理设计有效的连接池管理策略，使WEB应用程序数据库连接得到高效安全的复用，大大提高了系统的稳定性和访问效率.

1　传统数据库连接模式的分析

在使用 JAVA 开发基于数据库的WEB程序时，传统的模式基本是按以下步骤进行：1）在主程序(如Servnet \ Beans) 中建立数据库连接；2）进行SQL操作取出数据；3）断开数据库连接[1].

基于传统的数据库连接模式通常采用以下 2 种管理策略：1）为每个连接请求建立一个新的数据库连接；2）在系统的初始化时建立一个连接，所有对数据库的通信都通过此连接进行. 第一种情况的实现较简单，在任何时候都可以处理数据库的多用户并发请求，但问题是当有大量的用户同时登陆站点时，每个执行 JSP 实例的线程都将创建自己的数据库并直接与 RDBMS 引擎通信，使用结束后该线程将释放各自的连接. 第二种情况虽然可以消除这种并发的问题，但由于所有执行 JSP 实例的线程都使用同一个物理数据库连接，所有通过此连接的请求都被串行执行，因此当多个并发请求到达服务器时，只有一个请求被处理，其余请求则被低层次的套接字和网络协议封锁，导致过载时系统资源耗尽.

2　数据库连接池的设计

数据库连接池设计的基本思想是预先建立一些数据库连接的对象并放置内存中以备使用. 当程序中需要建立数据库连接时，只需从内存对象(连接池)中取出，同样使用完毕后，只需放回内存即可. 数据库连接的建立、使用和断开都由连接池自身管理，同时根据具体需要可以通过设置连接池的参数来控制连接池中的连接数、每个连接的最大使用次数等.

数据库连接池分为线程池、连接池、数据库操作 3 部分. 线程池统一对要执行的任务进行合理的线程分配和调度，执行用户的任务,管理计算机的线程资源. 连接池负责管理数据库连接的建立、释放和调度，首先在应用程序中先建立若干个连接，放置在内存对象中，当有数据库访问请求时，不需要执行连接数据库的操作，只需从连接池的空闲队列中取用连接，数据库访问完成后，将连接放回连接池中，供其他数据库操作时复用连接池中的连接. 数据库操作负责SQL 语句的执行、结果的返回，保证事务的完整性和异常处理[2]. 本文通过建立一个数据库连接池以及一套连接使用管理策略，提出了一个合理、有效的基于对象的数据库连接池的设计与实现，避免频繁地进行数据库的操作.

3　数据库连接池的实现

本文设计的数据库连接池主要包括连接池类（DBConnectionPool）、连接池管理类（DBConnetionPoolManager）和连接池调用类 USEConnectionPool[3]. 连接池类是对某一数据库所有连接的“缓冲池”，主要实现以下功能：1）从连接池获取或创建可用连接；2）使用完毕之后，把连接返还给连接池；3）在系统关闭前，断开所有连接并释放连接占用的系统资源；4）处理无效连接，以及限制连接池中的连接数量.

连接池管理类是连接池类的外覆类，可采用单例模式实现，即系统中只能有一个连接池管理

类的实例[4]. 其主要用于对多个连接池对象的管理，具体实现以下功能：1）装载并注册特定数据库的JDBC驱动程序；2）根据属性文件给定的信息，创建连接池对象；3）为方便管理多个连接池对象，为每一个连接池对象取一个名字，实现连接池名字与其实例之间的映射；4）跟踪客户使用连接情况，以便需要时关闭连接释放资源.

连接池调用类是数据库连接池的引用类[5]，主要实现以下功能：1）应用init()方法保存调用DBConnectionManager.getInstance()所返回的引用实例变量connMgr. 2）使用service()方法，调用连接池类的getConnection()建立一个连接，并执行相关数据库操作，最后用freeConnection()将连接返回给连接池. 3）使用destroy()方法，调用连接池管理类的release()方法关闭所有连接，释放所有资源.

由于整个连接池的程序比较长，限于篇幅，下面给出连接池实现类中主要代码.

连接池管理类DBConnectionManager

```
//管理类DBConnectionManager支持对一个或多个由属性文件定义的数据库连接池的访问
public class DBConnectionManager {
    static private DBConnectionManager instance; // 唯一实例
    private Hashtable pools = new Hashtable();
    . . . . . .
//将连接对象返回给由名字指定的连接池
    public void freeConnection(String name, Connection con) {
        DBConnectionPool pool = (DBConnectionPool) pools.get(name);
        if (pool != null) {
        pool.freeConnection(con);
        }
       }
//获得一个可用连接
   public Connection getConnection(String name, long time) {
       DBConnectionPool pool = (DBConnectionPool) pools.get(name);
       if (pool != null) {
       return pool.getConnection(time);
       }
       return null;
      }
 //根据指定属性创建连接池实例
   private void createPools(Properties props) {
          Enumeration propNames = props.propertyNames();
          while (propNames.hasMoreElements()) {
          . . . . . .
      }
```

连接池中连接类DBConnectionPool

```
class DBConnectionPool {
```

```
    private Vector freeConnections = new Vector();
    private int maxConn;
    . . . . . .
    //将不再使用的连接返回给连接池
  public synchronized void freeConnection(Connection con) {
    freeConnections.addElement(con);
    checkedOut--;
    notifyAll();
    }
    //从连接池获得一个可用连接
  public synchronized Connection getConnection() {
    Connection con = null;
    if (freeConnections.size() > 0) {
    con = (Connection) freeConnections.firstElement();
    freeConnections.removeElementAt(0);
    . . . . . .
     }
     return con;
    }
   }
```

连接池中调用类USEConnectionPool

```
  public class USEConnectionPool extends HttpServlet {
    private DBConnectionManager connMgr;
  //调用DBConnectionManager.getInstance()所返回的引用connMgr
    public void init(ServletConfig conf) throws ServletException {
    super.init(conf);
    connMgr = DBConnectionManager.getInstance();
    }
   //使用service()方法，调用连接池类的getConnection()建立一个连接，并执行相关数据库操作，
最后用freeConnection()将连接返回给连接池.
    public void service(HttpServletRequest req, HttpServletResponse res)throws IOException {
    res.setContentType("text/html");
    . . . . . .
    connMgr.freeConnection("idb", con);
    }
  //使用destroy()方法，调用连接池管理类的release()方法关闭所有连接，释放所有资源.
    public void destroy() {
    connMgr.release();
```

```
    super.destroy();
    }
  }
```

4 结束语

应用系统需要保证大量并发访问数据库时，采用数据连接池管理用户的连接是一种高效的连接控制策略，既可限制用户最大连接数，又可以动态保留若干空闲连接，使连接数据库的请求立即得到一个稳定的连接对象，极大地减少了连接和关闭数据库的操作，显著地提高了系统性能，使有限的计算机系统资源能为更多的用户提供更好的服务，保证用户的响应速度和服务质量，它是一种有效的解决方案.

参考文献：

[1] 光军，胡波. JSP 应用开发实例详解[M]. 北京：北京航天航空大学出版社，2002.

[2] 王晓路，卢建军，马莉. 基于 JAVA 的连接池优化 WEB 数据库连接[J]. 西安电子科技大学学报, 2005, 25(2): 228-231.

[3] 阎宏. JAVA 设计模式[M]. 北京：电子工业出版社, 2002.

[4] 牛志奇，丁天，田蕴哲. J2EE 核心模式[M]. 北京：机械工业出版社，2002.

[5] 刁磊，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, 39(30): 203-205.