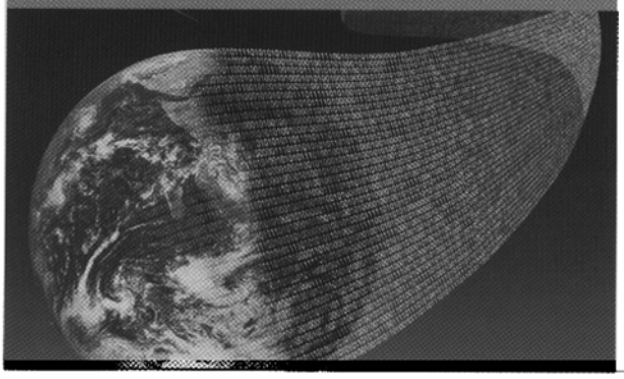
毕利 毕茹 李光明（宁夏大学数学计算机学院 750021）



数据库连接池在Oracle数据库中的实现

The Application of Database Connection Pool in Oracle

摘　要：在基于JDBC的数据库应用开发中，数据库连接的管理是一个难点，因为它是决定该应用性能的一个重要因素。本文在对数据库连接池技术分析的基础上，提出并实现了一个基于设计模式的解决方案和部分实例。

关键词：数据库 数据库连接池 JDBC

1 引言

在传统的两层结构中，客户端程序在启动时打开数据库连接，在退出程序时关闭数据库连接。这样，在整个程序运行中，每个客户端始终占用一个数据库连接，即使在大量没有数据库操作的空闲时间，如用户输入数据时，从而造成数据库连接的使用效率低下。

在三层结构模式中，数据库连接通过中间层的连接池管理。只有当用户真正需要进行数据库操作时，中间层才从连接池申请一个连接，数据库操作完毕，连接立即释放到连接池中，以供其他用户使用。这样，不仅大大提高了数据库连接的使用效率，使得大量用户可以共享较少的数据库连接，而且省去了建立连接的时间。

2 数据库连接池技术浅析

顾名思义，连接池最基本的思想就是预先建立一些连接放置于内存对象中以备使用：

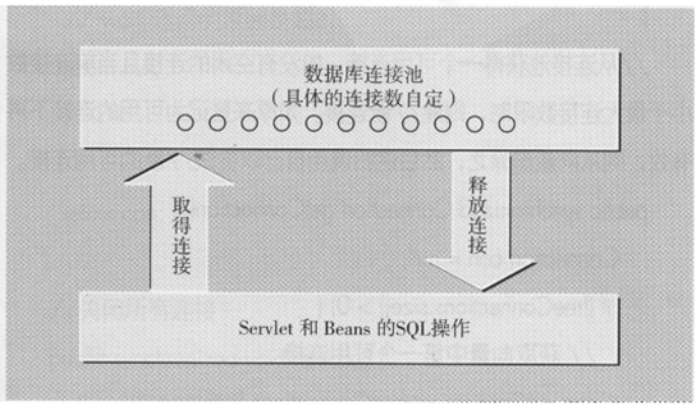


图 1　数据库连接池的工作模式

如图1所示，当程序中需要建立数据库连接时，只须从内存中取一个来用而不用新建。同样，使用完毕后，只需放回内存即可。而连接的建立、断开都有连接池自身来管理。同时，我们还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等。通过使用连接池，将大大提高程序效率，同时，我们可以通过其自身的管理机制来监视数据库连接的数量、使用情况等。

3 利用JDBC编写Oracle数据库的数据库连接池应用程序

数据库连接池是一个有价值的资源。获取数据库连接是一项耗时的工作，而且连接数非常有限。enterprise bean可从池中迅速获取连接。在bean释放连接之后可为其他bean使用。下面以本人编写的应用于Oracle数据库的一个数据库连接池程序为例，谈谈具体的实现方法。

3.1 Class DBConnectionManager 建立数据库连接池的管理类

public class DBConnectionManager implements Serializable 定义类

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，客户程序可以调用getInstance()方法访问本类的唯一实例。

3.2 Constructor Detail 具体构造

DBConnectionManager

private DBConnectionManager()

建构函数私有以防止其它对象创建本类实例

3.3 Method Detail 具体方法

（1）GetInstance

public static synchronized DBConnectionManager getInstance()

返回唯一实例。如果是第一次调用此方法，则创建实例。

Returns:

DBConnectionManager 唯一实例

（2）freeConnection

public void freeConnection(String name,Connection con)

将连接对象返回给由名字指定的连接池。

Parameters:

name - 在属性文件中定义的连接池名字

con - 连接对象

（3）getConnection

public Connection getConnection(String name)

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。

Parameters:

name - 在属性文件中定义的连接池名字

Returns:

Connection -可用的连接或null

（4）getConnection

public Connection getConnection(String name, long time)

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。

Parameters:

name - 连接池名字

time - 以毫秒计的等待时间

Returns:

Connection 可用连接或null

（5）release

public synchronized void release()

关闭所有连接，撤销驱动程序的注册。

（6）createPools

private void createPools(Properties props)

根据指定属性创建连接池实例。

Parameters:

props - 连接池属性

（7）init

private void init()

读取属性完成初始化。

（8）loadDrivers

private void loadDrivers(Properties props)

装载和注册所有JDBC驱动程序。

Parameters:

props - 属性

（9）log

private void log(String msg)

将文本信息写入日志文件。

private void log(Throwable e，String msg)

将文本信息与异常写入日志文件。

4 实例程序

由于整个连接池程序过长，现只将创建新的连接池的DBConnectionPool 的具体实现过程给出：

```
// 创建新的连接池
public DBConnectionPool(String name, String URL, String user, String password,
 int maxConn) {
this.name = name;   //连接池名字
this.URL = URL;   //数据库的JDBC URL
this.user = user;   //数据库的JDBC URL
this.password = password;   //密码或null
this.maxConn = maxConn;  //此连接池允许建立的最大连接数
}
//将不再使用的连接返回给连接池
public synchronized void freeConnection(Connection con) { // con 客户程序释放的连接
// 将指定连接加入到向量末尾
  freeConnections.addElement(con);
  checkedOut-;
  notifyAll();
}
//从连接池获得一个可用连接，如没有空闲的连接且当前连接数小于最大连接数限制，则建//新连接。如原来登记为可用的连接不再有效，则从向量删除之，然后递归调用自己以尝试//新的可用连接。
public synchronized Connection getConnection() {
  Connection con = null;
  if (freeConnections.size() > 0) {
   // 获取向量中第一个可用连接
   con = (Connection) freeConnections.firstElement();
freeConnections.removeElementAt(0);
```

```
try {
  if (con.isClosed()) {
    log("从连接池" + name+"删除一个无效连接");
    // 递归调用自己,尝试再次获取可用连接
    con = getConnection();
  }
}catch (SQLException e) {
    log("从连接池" + name+"删除一个无效连接");
    // 递归调用自己,尝试再次获取可用连接
    con = getConnection();
  }
}else if (maxConn == 0 || checkedOut < maxConn) {
  con = newConnection();
}
if (con != null) {
  checkedOut++;
}
return con;
}
//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。
public synchronized Connection getConnection(long timeout) {
// timeout 以毫秒计的等待时间限制
long startTime = new Date().getTime();
Connection con;
while ((con = getConnection()) == null) {
try {
  wait(timeout);
}catch (InterruptedException e) {}
  if ((new Date().getTime() - startTime) >= timeout) {
  // wait()返回的原因是超时
  return null;
  }
}
  return con;
}
//关闭所有连接
public synchronized void release() {
  Enumeration allConnections = freeConnections.elements();
  while (allConnections.hasMoreElements()) {
    Connection con = (Connection) allConnections.nextElement();
    try {
      con.close();
      log("关闭连接池" + name+"中的一个连接");
    }catch (SQLException e) {
      log(e, "无法关闭连接池" + name+"中的连接");
      }
    }
    freeConnections.removeAllElements();
  }
//创建新的连接
private Connection newConnection() {
    Connection con = null;
    try {
      if (user == null) {
      con = DriverManager.getConnection(URL);
    }else {
      con = DriverManager.getConnection(URL, user, password);
    }
    log("连接池" + name+"创建一个新的连接");
}catch (SQLException e) {
    log(e, "无法创建下列URL的连接: " + URL);
return null;
}0
return con;
}
}
```

5 结束语

通过上面的介绍，我们可以看出，连接池技术的关键就是其自身的管理机制。在本文给出的连接管理框架中，使用了一些广泛使用的设计模式，使得高效、安全的复用数据库连接成为可能。当然，还有一些问题没有考虑到，比如：没有实现对不同种类的数据库的联合管理等。以上的管理流程只是本人一点见解，关键是想向大家介绍一种思路，在此基础上，您可以进一步完善连接池技术为您所用。