基于 Hashtable 的数据库操作方法探讨*

游 中 胜
(重庆师范大学 数学与计算机科学学院, 重庆 400047)

摘 要 数据库查询操作是 Web 数据库应用程序开发中常见的一种操作, 传统的处理方式是在 Web 应用程序(如 JSP)中采用 java. sql 包中提供的操作数据库的类来操作数据库对象,但这种操作在资源未释放的情况下有可能导致一些严重的例外产生。通过设计 Hashtable 来作为 Web 应用程序和数据库结果集的一个中间层,然后在 Web 应用程序中解析 Hashtable,从而回避了数据库对象在 WEB 应用程序中的操作及其可能产生的其它问题。

关键词 :Hashtable ;Java 数据库
中图分类号 :TP311. 131 **文献标识码** :A **文章编号** :1672-6693(2006)01-0019-03

Operating Methods of Accessing Database Based on Hashtable

YOU Zhong-sheng
(College of Mathematics and Computer Science, Chongqing Normal University, Chongqing 400047, China)

Abstract :Web searching is a common operation in application sequence for web database. The traditional method is that in web application,use java. sql package which provides class to operate dababase. But this way may create vital error because at the moment the resources has not been released. Employ hashtable as a middle layer for the result of web and data application,then in the web application sequence resolve hashtable these problems can be avoided.

Key words :hashtable ;Java ;database

1 在 Web 应用程序中传统数据库操作方法[1~3]

1.1 传统的 Web 应用程序中数据库操作步骤

在 Web 应用程序中通常采用 java. sql 包中几个操作数据库的类(比如,连接类 Connection,语句类 Statement,结果集类 Resultset)来操作数据库对象,这个过程大致分为以下 5 个步骤 :(1)加载数据驱动程序,获取连接对象 Connection ;(2)通过连接对象创建执行 SQL 的语句对象 Statement ;(3)通过语句对象执行 SQL 语句,返回结果集 Resultset ;(4)对结果集进行解析,显示查询结果 ;(5)关闭结果集对象,语句对象,连接对象。其流程图如图 1 所示。

1.2 优点及不足之处

传统的 Web 应用程序执行数据库操作,优点在于无论数据量的多少,皆可适用。但该法必须执行上述 5 个步骤,在数据量不太大的情况下有如下不足 :(1)每次操作都必须按照上述步骤重复进行,增加了重复代码的编码量,使得代码繁杂,不简洁,延长了系统的开发周期,同时为代码的日常维护工作带来了麻烦 ;(2)如果每次在打开连接并处理完数据之后,没有关闭相应的数据库对象(由于没有进行这一步操作在短期内不会产生故障,所以往往忽视这一步),当连接数超过数据库的最大允许连接数的时候,将会产生 java. lang. NullPointer Exception 例外,这不仅导致系统不能正常运行,而且必须重新启动 Web 服务器甚至重新启动数据库才能使系统恢复正常。

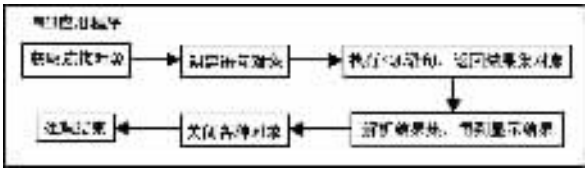

图 1 在 Web 应用程序中访问数据库的流程图

* 收稿日期 :2005-09-01 修回日期 :2005-12-05
作者简介 :游中胜(1970-),男,四川广安人,讲师,硕士研究生,研究方向为数据库应用与计算机网络。

2 基于 Hashtable 的数据库操作方法

2.1 引入中间层的优点

针对上述传统处理方法的不足,这里引入 Hashtable 作为中间层来访问数据库,即 Hashtable 作为 Web 应用程序和数据库对象之间的桥梁,则既可以避免上述情况的发生,还具有如下优点。

(1)结果集对象 Resultset 在 Web 应用程序中的解析,定位比较复杂,而如果使用 Hashtable,则可以用其中内置的一些非常方便、高效的方法来实现这些复杂的操作,如分页以及定位显示;

(2)使用 Hashtable,结构简单,逻辑性强,代码易读;

(3)在 Web 应用程序中直接操作 Hashtable,在源代码上和数据库对象没有任何联系,减少了一大堆对数据库对象的繁琐操作;

(4)在中间层 Hashtable 的 JAVABEAN 中封装了从连接数据库到返回 Hashtable 直至关闭所有数据库对象的相关操作,在 Web 应用程序中只需要传入一个 SQL 语句即可得到相应的保存有该 SQL 语句查询结果的 Hashtable,同时释放所有数据库资源,大大简化了操作;

(5)由于在 Web 应用程序中不需要操作数据库对象,所以就不存在关闭数据对象释放数据库资源的问题;

(6)由于按照一定的规则将结果集解析之后放入 Hashtable,所以对查询结果分页以及数据的定位有很大的便利。

2.2 使用中间层后的程序流程图

引入中间层(Hashtable)后,Web 应用程序的处理流程进行了某些变化,见图2所示。

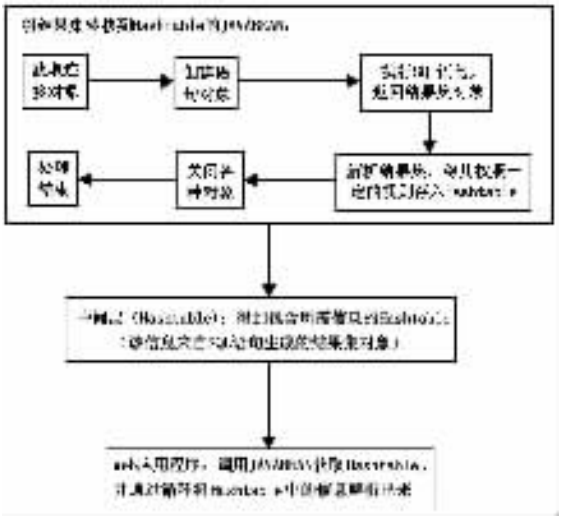

图2 引入中间层后的程序流图

2.3 结果集在 Hashtable 中的存储结构

假设结果集中各字段及记录数据见表1,在 Hashtable 中其对应的存储结构见表2。

表1 结果集存储结构

序号	Field_1	Field_2	…	Field_n
1	Field_1_1_value	Field_2_1_value	…	Field_n_1_value
2	Field_1_2_value	Field_2_2_value	…	Field_n_2_value
3	Field_1_3_value	Field_2_3_value	…	Field_n_3_value
…	…	…	…	…
M	Field_1_m_value	Field_2_m_value	…	Field_n_m_value

表2 结果集在 Hashtable 中的存储结构

序号	Field_1	Field_2	…	Field_n
1	(Field_1_1,Field_1_1_value)	(Field_2_1,Field_2_1_value)	…	(Field_n_1,Field_n_1_value)
2	(Field_1_2,Field_1_2_value)	(Field_2_2,Field_2_2_value)	…	(Field_n_2,Field_n_2_value)
3	(Field_1_3,Field_1_3_value)	(Field_2_3,Field_2_3_value)	…	(Field_n_3,Field_n_3_value)
…	…	…	…	…
M	(Field_1_m,Field_1_m_value)	(Field_2_m,Field_m_1_value)	…	(Field_n_m,Field_n_m_value)

3 在 Web 应用程序中解析 Hashtable 的实现过程[4~6]

3.1 返回 Hashtable 的 JAVABEAN

以下程序段先加载数据库驱动程序,并获取连接对象,然后通过连接对象创建语句对象,通过定义 getValue()函数返回特定的保存了结果集的 Hashtable;再通过定义 getFieldOfTable()函数,来返回 SQL 语句中包含的字段名。

```
public class Conversion {
  Statement stmt = null;
  String sqlQuery = "";//要执行的 SQL 语句
  public Conversion(String sqlQuery) {
    this. sqlQuery = sqlQuery;
      try {
        Class. forName("sun. jdbc. odbc. JdbcOdbcDriver");
      } catch(ClassNotFoundException e) {System. out. println
```

```
(e. getMessage());
        }
        conn = DriverManager. getConnection("jdbc:odbc:pretest","","");
        stmt = conn. createStatement();
    }
//返回特定的 Hashtable
    public java. util. Hashtable getValue() {
      int resLen = 0;String fieldName = "",fieldValue = "";
      Hashtable hashtable = new Hashtable();
      Vector vector = new Vector();vector. clear();
      ResultSet res = null;hashtable. clear();
      try {
        res = stmt. executeQuery(sqlQuery);
        vector = getFieldOfTable();int n = 0;
        while (res. next()) {
          for (int i = 0;i < vector. size();i + +) {
            fieldName = vector. elementAt(i). toString() + n;
            fieldValue = res. getString(vector. elementAt(i). toString());
            if (fieldValue = = null) fieldValue = "null";
            hashtable. put(fieldName, fieldValue);
          } n + +;}
        resLen = n;hashtable. put("resLen",new Integer(n));
        res. close();
      }
      catch (SQLException ex) {
        Res. close();Stmt. close();Conn. close;//出现例外关闭数据库对象
      }
      Res. close();Stmt. close();Conn. close;//正常关闭数据库对象
      return hashtable;
    }
// 根据 SQL 语句返回该 SQL 语句中包含的字段名
    public java. util. Vector getFieldOfTable() {
      Vector vectorField = new Vector();
      ResultSet rs = null;
      vectorField. clear();
      try {
        rs = stmt. executeQuery(sqlQuery);
        ResultSetMetaData rsmd = rs. getMetaData();
        int columns = rsmd. getColumnCount();
        if (vectorField. isEmpty() = = false) vectorField. clear();
        for (int i = 1;i < = columns;i + +) {
          vectorField. add(rsmd. getColumnName(i));
        }}
      catch (SQLException ex) {Rs. close();}
      return vectorField;
    }}
```

3.2　在 Web 应用程序中显示查询结果

利用 SQL 的查询语句及调用前面的 getValue() 函数返回包含了结果集的 Hashtable,然后在 Web 页中解析 Hashtable,以输出查询内容。

```
<%
Conversion conversion = new Conversion("select * from tablename");
Java. util. Hashtable hashtable = new Java. util. Hashtable();
Hashtable = conversion. getValue();
for(int i = 0;i < hashtable. size;i + +) {
    out. println(hashtable. get("Field_1" + i));
    out. println(hashtable. get("Field_2" + i));
    ….
    out. println(hashtable. get("Field_n" + i));
  }
% >
```

4　结束语

使用 Hashtable 作为中间层,将数据库表中的数据保存到 Hashtable 中,这在数据量不太大的时候是非常实用的。但是当数据表中的记录变得非常庞大的时候,使用这种方法比较消耗系统资源。所以需要根据数据量的实际情况采用相应的处理方法。

参考文献:

[1] 王珊,陈红. 数据库系统原理教程[M]. 北京:清华大学出版社,2003.

[2] 方晓华. 基于移动数据库的异构数据一致性技术研究[J]. 重庆师范大学学报(自然科学版),2003,20(4):24-27.

[3] 李建中. 数据库系统原理[M]. 第 2 版. 北京:电子工业出版社,2004.

[4] BRUCE E. Thinking in Java[M]. 北京:机械工业出版社,2004.

[5] HARVEY M,DEITEL P J. Java 大学教程[M]. 第 4 版. 北京:电子工业出版社,2002.

[6] 印旻. Java 语言与面向对象程序设计[M]. 北京:清华大学出版社,1999.

(责任编辑　黄　颖)