基于 JDBC 的数据库连接池及实现

Database Connection – Pool and it's Implementation Based on JDBC

王秀义 （山西工业职业技术学院 037003）

摘要：本文介绍了基于 JDBC 的数据库连接池的工作原理，提出了一个高效的连接使用管理策略，并给出了数据库应用的具体实现。

关键词：JDBC 连接池 引用记数 事务处理 封装

1 引言

Internet 的飞速发展，使机关、学校、公司、企业都已经建立或正在建立自己的网站。在开发基于数据库的 WEB 应用程序时，传统模式是在主程序（如 Servlet、Beans）中建立数据库连接，然后进行 SQL 操作取出数据，最后断开数据库连接。对于动态 Web 站点，往往是使用数据库存储的信息生成 Web 页面，每一个页面请求将导致一次数据库访问。连接数据库不仅要开销一定的通信和内存资源，还必须完成用户验证、安全上下文配置这类任务，因而往往成为最为耗时的操作，甚至可能会降低数据库服务器的性能。另外，用传统的模式，你必须去管理每一个连接，确保他们能被正确关闭，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，最终我们将不得不重启数据库。设计一个能经得起考验的 Web 数据库应用程序的最大挑战，在于如何合理地管理数据库连接。本文通过建立一个数据库连接池以及一套高效的连接使用管理策略，使得一个数据库连接可以得到高效安全的复用，大大地提高了系统效率。

2 JDBC 概述

JDBC（Java DataBase Connectivity，Java 数据库连接）是一种用于执行 SQL 的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。JDBC 提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序。

2.1 JDBC 驱动程序

JDBC API 提供两种主要接口：一是面向开发人员的 java. sql 程序包，使得 Java 程序员能够进行数据库连接，执行 SQL 查询，并得到结果集合。另一是面向底层数据库厂商的 JDBC 驱动程序，它介于前端应用程序与后端数据源之间，依其特性的不同共分为 4 种类型：

（1）JDBC – ODBC 桥 API 驱动程序。应用程序通过 JDBC – ODBC 桥（Bridge），以调用 ODBC 连接数据源，由于 Microsoft Windows 操作系统中的 ODBC 大多已支持各种类型的数据源，因此在建构上较为方便，可直接使用 JDK 所附的驱动程序（sun. jdbc. odbc. JdbcOdbcDriver）进行连接。

但由于必须经过桥的转换，因此在效率上并不十分理想，而且由于 ODBC 是以 C 语言编写而成，一有错误便是严重错误而非 Java 的 SQLException，使得 Java 程序无法控制这个错误而继续正常执行。

由于是经过 JDBC – ODBC 桥，因此服务器端需设置系统的 ODBC，但是一旦服务器端有所更动时，此 JDBC – ODBC 必须重新设置，在移植性上也不十分理想，因此，该类型驱动程序一般用于测试阶段，不太适合于企业应用上。

（2）Native API 驱动程序（非完全 Java）。此类型驱动程序也需经过类似桥的机制连接数据源，所不同的是，此类型的桥为原生函数库，是软件厂商针对其数据库（如：Oracle、Sybase、Informix 等）自行开发的，由于使用原生码，提高了执行速度，但出现的任何错误都可能使服务器死机。

（3）Net – Protocol 驱动程序（完全 Java）。此类

JDBC 驱动程序会将 SQL Statement 转换成为标准网络协议，交由数据库网关或应用程序服务器处理，应用程序服务器将标准网络协议翻译成为数据库厂商的专有特殊数据库访问协议与数据库通信。由于此类 JDBC 驱动程序常搭配应用程序服务器，因此可充分运用应用程序服务器的安全机制。另外由于是以 Java 开发的，因此不需额外的 ODBC 或原生函数库等机制，因此可移植性强。

（4）Native－Protocol 驱动程序（完全 Java）。此类 JDBC 驱动程序不需通过任何中介机制，直接转换 JDBC 调用，成为 DBMS 的网络协议，直接访问数据源。这对 Intranet 应用是高效的，可是数据库厂商的协议可能不被防火墙支持，缺乏防火墙支持在 Internet 应用中会存在潜在的安全隐患。

在实际应用中，考虑到效率和可移植性，尽量采用类型 3 或类型 4 驱动程序。

2.2 JDBC 流程

不论何种类型的 JDBC 驱动程序，JDBC 访问数据库的流程如下：

（1）载入 JDBC 驱动程序。要连接数据库，首先要加载 JDBC 驱动程序。标准方法如下：

Class. forName(<JDBC Driver Class Name>);

（2）建立与数据源的连接。成功加载 JDBC 驱动程序后，需要与数据源建立连接。

JDBC 提供了 Connection 与 DriverManager 类以建立数据源的连接，其语法为：

Connection con;

con = DriverManager. getConnection(<URL>);

con = DriverManager. getConnection (<URL>, (Properties>);

con = DriverManager. getConnection (<URL>, <User>, <Password>);

其中：

URL 用以定义连接数据源的协议，其定义方式如下：

<protocol>：<subprotocol >：<subname>

protocol：即 jdbc

subprotocol：依不同类型的 JDBC 驱动程序而有所不同。

Subname：数据源名称，同样依 JDBC 驱动程序而有所差异。

（3）执行 SQL 命令。成功建立与数据源的连接之后，便可执行 SQL 命令。任何一种类型的 SQL 命令。在 Java 中，当建立数据源连接之后，便需要建立 Statement 对象，以便执行 SQL 命令。Statement 对象依不同的 SQL 命令需求，提供了不同的方法。

（4）PreparedStatement。Statement 对象能执行 SQL 相关命令，但其前提是此 SQL 命令必须是明确定义且不能变动的，一旦 SQL 中的条件有所更改，则需重新定义。

为了能使同一 SQL 命令在条件更改之后仍能重复使用，JDBC 提供了 PreparedStatement 对象。PreparedStatement 的优点是一旦声明之后，便将 SQL 命令预交由数据源编译，当 PreparedStatement 正式被执行时，仅需传入所需的参数，数据源不需重新编译此 SQL 命令而直接执行。而当条件参数改变时，也仅需重新传入参数便能直接执行预先被编译好的 SQL 命令。

（5）ResultSet 与 ResultSetMetaData。当以 executeQuery 方法执行 SELECT 命令时，其结果会以 ResultSet 对象保存。ResultSet 提供了 get 与 update 方法，用来取得与更新 ResultSet 的数据。另外，要取得数据表中字段的数目、类型及属性，可使用 ResultSet 的 getMetaData 方法，其返回的结果会以 ResultSetMetaData 对象保存。

（6）关闭 ResultSet 及 Connection。使用 JDBC 的最后一个步骤是关闭所有的 JDBC 对象及与数据源的连接。

3 连接池原理

对于共享资源，有一个很著名的设计模式：资源池。该模式正是为了解决资源频繁分配、释放所造成的问题的。把该模式应用到数据库连接管理领域，就是建立一个数据库连接池，提供一套高效的连接分配、使用策略，最终目标是实现连接的高效、安全的复用。

3.1 建立连接池

首先要建立一个静态的连接池，在系统初始化时就分配好池中的连接，并且不能够随意关闭。使用 Java 中的容器类可以很方便地构建连接池，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，这样就可以避免连接随意建立、关闭所造成的系统开销。

3.2 分配、释放策略

建立好连接池后，还需要提供一套相应的分配、释放策略，来保证数据库连接的有效复用。

当客户请求数据库连接时，首先看连接池中是否有空闲连接，如果存在空闲连接则把连接分配给客户，并作相应处理，主要的处理策略就是标记该连接为已分配。若连接池中没有空闲连接，就在已经分配出去的连接中，寻找一个合适的连接给客户，此时该连接在多个客户间复用。

当客户释放数据库连接时，可以根据该连接是否被复用，进行不同的处理。如果连接没有使用者，就放入到连接池中，而不是被关闭。

3.3 配置策略

对于数据库连接池，一般的配置策略是根据具体的应用需求，给出一个初始的连接池中连接的数目以及一个连接池可以扩张到的最大连接数目。本方案就是按照这种策略实现的。

4 关键技术

4.1 引用记数

引用记数(Reference Counting)是一种对于复用资源广泛使用的设计模式。我们把该方法运用到对于连接的分配释放上。每一个数据库连接，保留一个引用记数，用来记录该连接的使用者的个数。具体实现方法是采用两级连接池－空闲池和使用池。空闲池中存放目前还没有分配出去被使用的连接，一旦一个连接被分配出去，它就会被放入到使用池中，并且增加引用记数。

采用引用记数可以高效地使用连接。假若空闲池中的连接已被全部分配出去，此时可以根据相应的策略从使用池中挑选出一个已经正在使用的连接用来复用，而不是随意拿出一个连接去复用。策略可以根据需要去选择，我们采用的策略是：复用引用记数最小的连接。Java 的面向对象特性，使得我们可以灵活的选择不同的策略。具体方法是：提供一个不同策略共用的抽象接口，各个具体的策略都实现这个接口，致使策略的处理逻辑与策略的实现逻辑相互分离。

4.2 事务处理

使用数据库连接进行普通的数据库访问是比较简单的。而在事务处理中，如果简单地采用上述的连接复用策略，就会发生问题，因为无法控制属于同一个事务的多个数据库操作方法的动作，可能这些数据库操作是在多个连接上进行的，并且这些连接可能被其他非事务方法复用。

Connection 提供了对事务操作的支持，可以通过设置 Connection 的 AutoCommit 属性为 false，显式的调用 commit 或者 rollback 方法来实现。但是，要安全、高效的进行连接复用，就必须提供相应的事务支持机制。我们采用的方法是：显式的事务支撑方法，每一个事务独占一个连接。该方法可以大大降低对于事务处理的复杂性，并且又不会妨碍连接的复用，因为隶属于该事务的所有数据库操作都是通过这一个连接完成的，并且事务方法又复用了其他一些数据库方法。

显式的事务开始、结束(commit 或者 rollback)声明及一个事务注册表是连接管理服务的主要内容。事务注册表用于登记事务发起者与事务使用的连接之间的对应关系，它是在运行过程中根据实际调用情况动态生成的，可将使用事务部分和连接管理部分隔离开来。事务使用的连接在该事务运行中不能被复用。在我们的实现中，用户标识是通过使用者所在的线程来标识的。后面的所有对于数据库的访问都是通过查找该注册表，使用已经分配的连接来完成的。当事务结束时，从注册表中删除相应表项。

4.3 封装

由于普通的数据库方法和事务方法对于连接的使用(分配、释放)是不同的，为了便于使用，对外提供一致的操作接口，我们利用 Java 的多态性对连接进行了封装：即普通连接和事务连接。普通连接和事务连接均实现了一个 DbConnection 接口，对于接口中定义的方法，分别根据各自的特点作了不同的实现，这样对于连接的处理就非常一致了。

4.4 并发问题

为了使我们的连接管理服务有更大的通用性，就必须要考虑到多线程环境，即并发问题。在一个多线程的环境下，必须要保证连接管理自身数据的一致性和连接内部数据的一致性，使用 Java 提供的 synchronized 关键字，就很容易使连接管理成为线程安全的。

5 具体实现

本文介绍的数据库连接池包括一个管理类 DBConnectionManager，负责提供与多个连接池对象（DBConnectionPool 类）之间的接口。每一个连接池对象管理一组 JDBC 连接对象，每一个连接对象可以被任意数量的 JSP 或 Servlet 共享。

5.1 类 DBConnectionPool

（1）功能

- 从连接池获取（或创建）可用连接。
- 把连接返回给连接池。
- 在系统关闭时释放所有资源，关闭所有连接。
- DBConnectionPool 类还能够处理无效连接（原来登记为可用的连接，由于某种原因不再可用，如超时，通信问题），并能够限制连接池中的连接总数不超过某个预定值。

（2）说明

name //连接池名字

URL //数据库的 JDBC URL

user //数据库帐号

password//密码

maxConn//连接池允许建立的最大连接数

public synchronized void freeConnection（Connection con）//将连接返回给连接池

public synchronized Connection getConnection（）//从连接池获得可用连接

public synchronized Connection getConnection（long timeout）//从连接池获取可用连接，并指定客户程序能够等待的最长时间。

private Connection newConnection（）//创建新连接

public synchronized void release（）//关闭所有连接。

5.2 管理类 DBConnectionManager

（1）功能

- 装载和注册 JDBC 驱动程序。
- 根据在属性文件中定义的属性创建连接池对象。
- 实现连接池名字与其实例之间的映射。
- 跟踪客户程序对连接池的引用，保证在最后一个客户程序结束时安全地关闭所有连接池。

（2）说明

getInstance（）方法//访问连接池的唯一实例

private DBConnectionManager（）//建构函数私有以防止其它对象创建本类实例

getResourceAsStream（）方法//用于定位并打开外部文件

db. properties//属性文件，它包含定义连接池的键－值对。

loadDrivers（）方法//实现装载和注册所有在 drivers 属性中指定的 JDBC 驱动程序

createPools（）//创建连接池对象。

Hashtable 类 //实现连接池名字到连接池对象之间的映射。

6 结语

本文提出的连接管理策略的核心思想是：连接复用。通过建立一个数据库连接池以及一套连接使用管理策略，使得一个数据库连接可以得到高效、安全的复用，避免了数据库连接频繁建立、关闭的开销。另外，由于对 JDBC 中的原始连接进行了封装，从而方便了数据库应用对于连接的使用（特别是对于事务处理），提高了开发效率，也正是因为这个封装层的存在，隔离了应用的本身的处理逻辑和具体数据库访问逻辑，使应用本身的复用成为可能。

参考文献

1 黄嘉辉，Java 网络程序设计[M]，清华大学出版社，2002。

2 杨绍方，Java 编程实用技术与案例[M]，清华大学出版社，2000。

3 http://www. researchorg. java