一种基于代理的 Java 数据库连接池设计与实现

Design and Realization by Proxy – based Java Database Connection – pool

杨 光　耿祥义　王 瑞　（大连交通大学软件学院 116000）

摘要：普通 Java 数据库连接池的缺陷是不能对其中的连接进行独占控制，本文所设计的 Java 数据库连接池不仅具有普通连接池的功能，而且能对其中的连接进行独占控制。

关键词：代理 连接池 JDBC

1 引言

JDBC（Java Database Connection，Java 数据库连接）是一组用于执行 SQL 语句的 Java API，由一些 Java 类和接口组成。由于 Java 语言简单、平台无关，JDBC 被广泛的应用于数据库应用系统的开发。在基于 JDBC 的应用系统中，建立数据库连接是非常消耗资源的操作，如果频繁的进行创建、关闭数据库连接操作会严重影响系统的性能，严重时可能使系统崩溃。为此，在很多的系统中都引入了数据库连接池模块。在连接池中预先创建了一些连接，应用程序每次需要进行数据库操作时，都从数据库连接池中取得连接，使用结束后再把连接归还给连接池。这样就避免了频繁创建、关闭数据库连接的开销。图 1 显示了应用程序使用数据库连接池访问数据库的模式。

在图 1 中，通过引入 Java 数据库连接池可以大大的提高系统的性能，但是同时也给用户使用数据库连接带来了不便。不同的连接池实现要求用户使用不同的方法获得连接，这是可以理解的。然而，普通的数据库连接池允许用户使用其中的连接，但却无法阻止用户关闭其所用的连接，这样一来，连接池必须频繁地检查哪些连接被用户关闭，需要将关闭的连接重新打开，从而增加了数据库连接池管理的复杂度，其原因是由于数据库连接池不能对其中的连接进行独占控制引起的。

本文采用 Java 的动态代理（proxy）技术有效的解决了上述问题，其关键技术是绑定连接对象到实现了 Connection 接口的一个动态代理实例。这样一来，用户调用连接的任何方法都被相应的代理实例转发给 InvocationHandler 实例处理，然后，在 InvocationHandler 实例中实现了对所调用方法的处理，从而实现了连接池对其中连接的独占性控制。

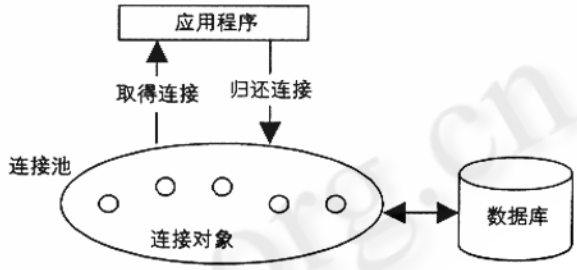

图 1 应用程序通过连接池访问数据库

2 Java 动态代理

Java 动态代理实现了类似于 Windows 钩子的功能，可以把接口方法的调用转发给 InvocationHandle 实例。要生成 Java 动态代理，只需要调用 java. lang. reflect. Proxy 类的实例方法：

```
static Object newProxyInstance( ClassLoader ldr,
  Class[ ] itfs,
  InvocationHandler handler
  )
Throws IllegalArgumentException;
```

该方法在内存中生成新的二进制类。这个特殊的类通过将单个方法转发到 InvocationHandler 实例，实现了 itfs 数组中指定的所有接口。然后由 ldr 指定的类加载器将新类加载到虚拟机，并用新类构造转发到

handler 的代理实例。InvocationHandler 实例是代理转发接口方法的目标对象，InvocationHandler 接口的定义如下：

```
package java.lang.refrect;
public class interface InvocationHandler {
    public Object invoke(Object proxy,
        Method method,
        Object[] args
        )
     throws Throwable;
}
```

在该接口中只有一个 invoke 方法，用来处理所有转发到 InvocationHandler 实例的接口方法。

3 JDBC 数据库连接池

本文设计的数据库连接池的主要创新点是：实现连接池对其中连接的独占性控制。为了实现连接池对其中连接的独占性控制，当用户从连接池中获取连接时，连接池会把其中的空闲连接对象与实现 Connection 接口的 Java 动态代理绑定，并把该代理对象返回给用户。用户使用代理对象调用 Connection 接口中的方法都被转发给实现了 InvocationHandler 接口的 ConnectionHandler 实例。ConnectionHandler 实例调用 invoke 方法实现了对 Connection 接口中方法的进一步处理。

3.1 连接对象与代理的绑定

为了实现连接池对其中连接的独占性控制，需要把连接对象都与动态代理的实例绑定，同时提供实现 InvocationHandler 接口的实例对象。动态代理实例将把方法的调用转发给 InvocationHandler 实例，由 InvocationHandler 实例调用 invoke 方法进行处理。

ConnectionHandler 类实现了上述功能。下面给出 ConnectionHandler 类的定义（伪码）：

```
class ConnectionHandler implements InvocationHandler
{//实现了 InvocationHandler 接口
    private Connection proxyedObj;//被绑定的连接对象
    public Connection bind(Connection conn) {//绑定连接对象与动态代理
        proxyedObj = conn;
        //返回实现了 Connection 接口的代理实例
        return (Connection) Proxy.newProxyInstance
(proxyedObj.getClass().getClassLoader(),
            proxyedObj.getClass().getInterfaces(),//Connection 接口
            this//InvocationHandler 实例对象
            );
    }
    public Object invoke(Object proxy, Method method, Object[] args)
        throws Throwable {
    Object result = null;
    if (调用的方法是 close) {
        连接池回收连接对象 proxyedObj
    } else {//不是 close 方法
        转发给 proxyedObj 对象，返回结果给 result
    }
    return result;
    }
}
```

3.2 连接池的创建

本文采用单例模式创建数据库连接池实例，这样可以避免在一个应用中有多个连接池实例对象存在。下面给出连接池类 DBConnectionPool 的定义（主要方法）：

```
public class DBConnectionPool {
    public static DBConnectionPool getInstance()//单例模式获取数据库连接池实例
    public Connection getConnection()//获取数据库连接
    public void release()//释放数据库连接池
}
```

调用 getInstance 方法创建的数据库连接池对象可以通过读取配置文件 properties.xml 与某个具体的数据库实例建立联系，并根据配置文件预置一定数量的连接对象，以备使用。下面是配置文件：

```
<properties version = "1.0" >
  <entry key = " DBDriver" > com. mysql. jdbc. Driver </entry >
  <entry key = " DBURL" > jdbc: mysql: ///test </entry >
  <entry key = " UserID" > root </entry >
  <entry key = " Password" > sundyangg </entry >
  <entry key = " MinSize" > 10 </entry >
  <entry key = " MaxSize" > 25 </entry >
</properties >
```

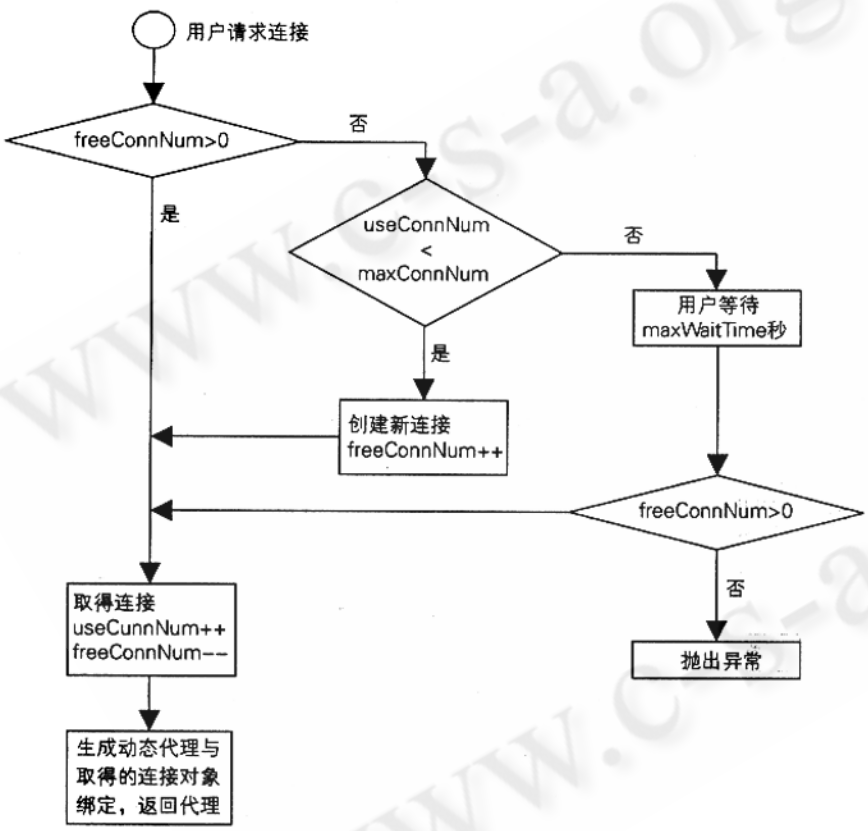

图 2　连接分配流程图

3.3 连接的分配与释放

Java 数据库连接池的核心功能就是对其中的连接进行高效、安全的管理。重点要解决的问题就是连接池中连接的分配和释放，它们对系统的性能有很大的影响。连接的合理分配、释放可以提高连接的复用率，降低系统由于频繁创建连接引起的开销，提高了系统的响应速度。下面给出具体方法。

当用户请求数据库连接时，查看连接池中是否有空闲的连接（freeConnNum），如果有就取得一个连接并与动态代理绑定，返回代理对象给用户；否则，查看目前池中连接的总数是否达到了最大连接数（maxConnNum），如果没有达到最大连接数，就创建一个新的连接，并把这个新的连接与动态代理绑定，返回代理对象给用户；否则，设置用户最大等待时间（maxWaitTime），用户进入等待状态，maxWaitTime 之后还没有空闲的连接，则向用户抛出异常。图 2 给出了连接分配的流程图。

为了方便连接池管理其中的连接，不允许用户释放连接，所有的连接都由连接池对象调用 release 方法进行释放。用户调用 close 方法，动态代理将把该方法转发给相应 ConnectionHandler 实例的 invoke 方法，在 invoke 方法中连接池将回收这个连接。如果用户调用的是其它的方法，invoke 方法将把方法调用转发给真实的连接对象。这样可以大大的降低连接池管理连接的复杂性，实现了对其中连接的独占性控制。

4 总结

本文使用 Java 动态代理技术设计实现的数据库连接池成功地解决了连接池不能对其中连接进行独占性控制的问题，同时也降低了连接池管理连接的复杂性。

参考文献

1 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，第 30 卷(9).
2 王秀义，基于 JDBC 的数据库连接池及实现，计算机系统应用(J)，2005，4.
3 (美)Stuart Dabbs Halloway. Java，平台组件开发，清华大学出版社(M)，三 2004，9.