Connection Pools

A web application that has been deployed on a server may have many clients accessing it. If each time the database connection is needed it has to be reopened, performance will degenerate quickly. The standard solution is to create a pool of connections that can be used when needed and then returned to the pool for use by other servlets.

The pool is stored in the ServletContext object so that all servlets in the application can use it to get connections. A special servlet called ConnectionServlet is needed to create the pool. It can also be used to display some information about the usage. The web deployment file, web.xml, contains along with the servlet name and class, <load-on-startup>1</load-on-startup>. This assures that the connection pool will be created before anything else is done.

The pool is implemented as a vector of ConnectionObjects. These objects not only contain the connection, but also some useful information about it. They record the number of times the connection was used, the last time it was accessed and whether or not it is currently in use.

The vector is stored in a class called ConnectionPool. This class has methods to create the connection pool, destroy it when no longer needed, get a connection from the pool and return a connection to the pool after it is no longer needed by the servlet that was using it. Some connection information is stored in a configuration file called ConnectionPool.cfg. This way the ConnectionPool class is not tied to any one database. Changing a few lines in the configuration file changes the target database.

Servlets that use the connection pool only require a little code in order to get the pool from the ServletContext and then obtain a connection. The main work of getting connections is done by the ConnectionPool class.

The code for this connection pool was adapted from that found in *Java Servlets, Developers Guide* by Karl Moss. Parts of it can be found on pages 226 to 249. The complete code is available on Karl Moss' website, http://www.servletguru.com.

The ConnectionObject

The ConnectionObject class stores a connection and some data related to it. It uses a Date object to store information about the last access time. The code follows:

```
package produce;

import java.sql.*;
import java.io.*;
// There is a Date object in both java.sql and java.util. The compiler must be told which one to use.
import java.util.Date;

public class ConnectionObject
{
    private Connection con;
    private int useCount;
    private Date lastAccessTime;
    private boolean inUse;

    public Connection getConnection () {return con;}
```

```
    public int getUseCount () {return useCount;}
    public Date getLastAccessTime () {return lastAccessTime;}
    public boolean getInUse () {return inUse;}

    public void setConnection (Connection c) {con = c;}
    public void setUseCount (int u) {useCount = u;}
    public void setLastAccessTime (Date d) {lastAccessTime = d;}
    public void setInUse (boolean i) {inUse = i;}

    public boolean isAvailable () {return !inUse;}

} // ConnectionObject
```

The ConnectionServlet Class

The ConnectionServlet class stores the connection pool in the ServletContext. It can then be accessed by any servlet in the web application. Like any servlet, it has either a doGet or a doPost method. But in this case, its main use is not to respond to a request. Instead it instantiates the connection pool and stores it in the ServletContext. Code for the init and destroy methods follow. The init method is automatically run when the servlet is loaded and the destroy method when the application ends.

```
    ConnectionPool pool;
    public static String PoolKey = "produce.ConnectionServlet";

    // The init method gets a new pool and initializes it.
    public void init (ServletConfig cfg)
    {
        try
        {
            super.init (cfg);
            pool = new ConnectionPool ();
            pool.initialize ();
        } catch (ServletException e) {System.out.println ("Could not initialize");}
        ServletContext context = getServletContext ();
        context.setAttribute (PoolKey, pool); // Store the pool in the ServletContext.
    } // init

    /* The destroy method closes all connections in the pool and removes the attribute from the
    ServletContext. */
    public void destroy ()
    {
        getServletContext ().removeAttribute (PoolKey);
        if (pool != null) pool.destroy ();
        super.destroy ();
    } // destroy
```

A Servlet that Uses the Pool

Before going into details about how the ConnectionPool class works, we can see just how a servlet uses the ServletContext to get a connection. The only code needed gets the pool from the ServletContext and then gets a connection.

```
ServletContext context = getServletContext ();
String key = ConnectionServlet.PoolKey;
Object object = context.getAttribute (key);
if (object == null) out.println ("No connection pool.");
else
{
    ConnectionPool pool = (ConnectionPool) object;
    Connection con = pool.getConnection ();
    // Use the connection to access the database.
    pool.close (con);
}
```

The rest of the servlet is the same as before, except that it no longer throws a ClassNotFoundException. That is handled by the ConnectionPool class.

The Configuration File

ConnectionPool.cfg contains some connection information needed to create the connection. The file can be easily changed to refer to a different database. This makes the ConnectionPool class more flexible.

```
# ConnectionPool.cfg
# Defines connection pool parameters

JDBCDriver=sun.jdbc.odbc.JdbcOdbcDriver
JDBCConnectionURL=jdbc:odbc:produce
ConnectionPoolSize=5
ConnectionPoolMax=50
ConnectionUseCount=5
```

The configuration file can be loaded into a Properties class and from there into the ConnectionPool class. Changing the JDBCDriver or the JDBCConnectionURL will change the target database.

The ConnectionPool Class

The ConnectionPool class is the most complicated. It not only creates the vector of ConnectionObjects, but it also has methods to manage these objects.

```
package produce;

import java.sql.*;
import java.io.*;
// There is a Date object in java.sql, so the specific Data object required is specifically imported here.
import java.util.Date;
import java.util.Vector;
import java.util.Properties;

/* The ConnectionPool class manages a pool of database connections. It reads configuration information
from ConnectionPool.cfg and uses this to create a connection to the database. It also has methods that get
a connection, get the entire pool, close a connection by returning it to the pool, and destroy the pool.
```

```
public class ConnectionPool
{
	private String JDBCDriver;
	private String JDBCConnectionURL;
	private int ConnectionPoolSize;
	private int ConnectionPoolMax;
	private int ConnectionUseCount;
	private Vector pool;

	// The initialize method is called by the init method of the ConnectionServlet class.
	public void initialize()
	{
		String config = "ConnectionPool.cfg";
		boolean loaded = loadConfig (config);
		if (!loaded) System.out.println ("Error loading config file.");
		else
		{
			pool = new Vector ();
			fillPool (ConnectionPoolSize);
		}
	} // initialize

	public Vector getConnectionPoolObjects () {return pool;}

	/* The fillPool method gets a connection and then checks to see how many connections the database
	can support.  If it is less than ConnectionPoolSize, the pool size is adjusted.  It then fills the vector
	with new ConnectionObjects. */
	private synchronized void fillPool (int poolSize)
	{
		try
		{
			Connection con; int count = 0;
			int maxConnections = 0;
			while (count < poolSize)
			{
				ConnectionObject conObject = new ConnectionObject ();
				Class.forName (JDBCDriver);
				con = DriverManager.getConnection (JDBCConnectionURL);
				conObject.setConnection (con);
				/* The first time through the loop, find out the maximum number of connections that
				the database can support. */
				if (count == 0)
				{
					con = conObject.getConnection ();
					DatabaseMetaData metaData = con.getMetaData ();
					maxConnections = metaData.getMaxConnections ();
					if (poolSize > maxConnections)
					{
						System.out.println ("Pool size too large.");
						poolSize = maxConnections;
					}
```

```
                }
                conObject.setInUse (false); // Mark the connection as available.
                conObject.setUseCount (0); // Start the use count off at 0.
                conObject.setLastAccessTime (new Date ()); // Set the first access time.
                pool.addElement (conObject); // Add the object to the vector.
                count ++;
            }
        } catch (SQLException e) {System.out.println ("SQLException");}
        catch (ClassNotFoundException e){System.out.println ("Class Not Found exception.\n");}
} // fillPool

// The method getConnection is synchronized so that only one servlet can get a connection at a time.
public synchronized Connection getConnection ()
{
        Connection con = null;
        ConnectionObject connectionObject = null, conObject = null;
        int poolSize = pool.size (), count = 0;
        boolean found = false;

        if (pool == null) return null;  // Do not access an empty pool.
        // Find the first available connection in the pool.
        while ((count < poolSize) && !found)
        {
                conObject = (ConnectionObject) pool.elementAt (count);
                if (conObject.isAvailable ()) found = true;
                else count ++;
        }
        if (found) connectionObject = conObject;

        if (connectionObject == null)        System.out.println ("All connections in use.");
        else
        {
                connectionObject.setInUse (true); // Make the connection unavailable to others.
                int useCount = connectionObject.getUseCount () + 1;
                connectionObject.setUseCount (useCount); // Increment the use count.
                connectionObject.setLastAccessTime (new Date ()); // Change the access date.
                con = connectionObject.getConnection ();
        }
        return con;
} // getConnection

// The close method makes the connection available again.  It does not really close the connection.
public synchronized void close (Connection con)
{
        int index = find (con);
        if (index != -1)
        {
                ConnectionObject conObject = (ConnectionObject) pool.elementAt (index);
                conObject.setInUse (false);
                conObject.setLastAccessTime (new Date ());
        }
```

```
        else System.out.println ("Connection not found in pool.");
    } // close

    /* find is a private method that searches through the vector to find a connection with the same
    catalog name.   When one is found, its index is returned to the close method.
    private int find (Connection con)
    {
        int index = 0;
        boolean found = false;
        try
        {
            ConnectionObject conObject;
            String catalog = con.getCatalog ();

            while ((index < pool.size ()) && !found)
            {
                conObject = (ConnectionObject) pool.elementAt (index);
                Connection poolCon = conObject.getConnection ();
                String name = poolCon.getCatalog ();
                if (catalog.equals (name)) found = true;
                else index ++;
            }
        } catch (SQLException e) {System.out.println ("Catalog Exception");}
        if (found) return index;
        else return -1;
    } // find

    /* The destroy method is executed when the application is finished.  It closes each connection in the
    pool and then sets the pool to null. */
    public void destroy ()
    {
        try
        {
            if (pool != null)
            {
                // Close each connection in the pool.
                for (int count = 0; count < pool.size(); count++)
                {
                    ConnectionObject co = (ConnectionObject) pool.elementAt(count);
                    Connection con = co.getConnection ();
                    con.close (); // This really closes the connection.
                }
            }
        } catch (SQLException e) {System.out.println ("Destroy error.");}
        pool = null;
    } // destroy

    /* The loadConfig method is used to read in the configuration file and store its data in the class
    variables. */
    private boolean loadConfig (String config)
    {
```

```
            InputStream in = null;
            boolean loaded = false;
            try
            {
                ClassLoader loader = getClass().getClassLoader(); // Get the loader for this class.
                if (loader != null) in = loader.getResourceAsStream(config);
                else in = ClassLoader.getSystemResourceAsStream(config);

                // If the input stream is null, then the configuration file was not found.
                if (in == null)
                    System.out.println ("ConnectionPool configuration file, '"+ config + "', not found");
                else
                {
                    Properties JDBCProperties = new Properties();
                    // Load the configuration file into the properties table
                    JDBCProperties.load(in);

                    JDBCDriver = JDBCProperties.getProperty ("JDBCDriver");
                    JDBCConnectionURL  = JDBCProperties.getProperty ("JDBCConnectionURL");
                    ConnectionPoolSize = Integer.parseInt
                        (JDBCProperties.getProperty ("ConnectionPoolSize"));
                    ConnectionPoolMax  = Integer.parseInt
                        (JDBCProperties.getProperty ("ConnectionPoolMax"));
                    ConnectionUseCount = Integer.parseInt
                        (JDBCProperties.getProperty ("ConnectionUseCount"));
                    loaded = true;
                }
            }   catch (IOException e) {System.out.println ("IOException");}
                finally {if (in != null) try {in.close();}catch (IOException ex) {}} // Close the input stream.
            return loaded;
        } // loadConfig

} // ConnectionPool
```

Reference

Karl Moss, *Java Servlets Developer's Guide*, chapter 9, McGraw-Hill/Osborne, 2002.