DBCP Hibernate

About Hibernate & DBCP

Hibernate 2 includes a limited DBCP ConnectionProvider. Not all features of DBCP were supported and with Hibernate3 this limited implementation was deprecated.

There is an implementation on this page. The original author said:

- The implementation below supports all DBCP features and is a drop in replacement for the default version.

However this is untrue. It fails to support:

- Almost all `hibernate.dbcp.ps.*` properties
- `hibernate.dbcp.whenExhaustedAction`
- ... maybe more?

so personally I feel rather dubious about using it.

This provider can be used on Hibernate 2 & 3 (with some import statements adjustments).

DBCPConnectionProvider

```
/*
 * Copyright 2004 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.hibernate.connection;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Iterator;
import java.util.Properties;

import org.apache.commons.dbcp.BasicDataSource;
import org.apache.commons.dbcp.BasicDataSourceFactory;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.HibernateException;
import org.hibernate.cfg.Environment;

/**
 * <p>A connection provider that uses an Apache commons DBCP connection pool.</p>
 *
 * <p>To use this connection provider set:<br>
 * <code>hibernate.connection.provider_class&amp;nbsp;org.hibernate.connection.DBCPConnectionProvider</code></p>
 *
 * <pre>Supported Hibernate properties:
 *   hibernate.connection.driver_class
 *   hibernate.connection.url
 *   hibernate.connection.username
 *   hibernate.connection.password
 *   hibernate.connection.isolation
 *   hibernate.connection.autocommit
 *   hibernate.connection.pool_size
 *   hibernate.connection (JDBC driver properties)</pre>
```

```
 * <br>
 * All DBCP properties are also supported by using the hibernate.dbcp prefix.
 * A complete list can be found on the DBCP configuration page:
 * <a href="http://jakarta.apache.org/commons/dbcp/configuration.html">http://jakarta.apache.org/commons/dbcp
/configuration.html</a>.
 * <br>
 * <pre>Example:
 *   hibernate.connection.provider_class org.hibernate.connection.DBCPConnectionProvider
 *   hibernate.connection.driver_class org.hsqldb.jdbcDriver
 *   hibernate.connection.username sa
 *   hibernate.connection.password
 *   hibernate.connection.url jdbc:hsqldb:test
 *   hibernate.connection.pool_size 20
 *   hibernate.dbcp.initialSize 10
 *   hibernate.dbcp.maxWait 3000
 *   hibernate.dbcp.validationQuery select 1 from dual</pre>
 *
 * <p>More information about configuring/using DBCP can be found on the
 * <a href="http://jakarta.apache.org/commons/dbcp/">DBCP website</a>.
 * There you will also find the DBCP wiki, mailing lists, issue tracking
 * and other support facilities</p>
 *
 * @see org.hibernate.connection.ConnectionProvider
 * @author Dirk Verbeeck
 */
public class DBCPConnectionProvider implements ConnectionProvider {

    private static final Log log = LogFactory.getLog(DBCPConnectionProvider.class);
    private static final String PREFIX = "hibernate.dbcp.";
    private BasicDataSource ds;

    // Old Environment property for backward-compatibility (property removed in Hibernate3)
    private static final String DBCP_PS_MAXACTIVE = "hibernate.dbcp.ps.maxActive";

    // Property doesn't exists in Hibernate2
    private static final String AUTOCOMMIT = "hibernate.connection.autocommit";

    public void configure(Properties props) throws HibernateException {
        try {
            log.debug("Configure DBCPConnectionProvider");

            // DBCP properties used to create the BasicDataSource
            Properties dbcpProperties = new Properties();

            // DriverClass & url
            String jdbcDriverClass = props.getProperty(Environment.DRIVER);
            String jdbcUrl = props.getProperty(Environment.URL);
            dbcpProperties.put("driverClassName", jdbcDriverClass);
            dbcpProperties.put("url", jdbcUrl);

            // Username / password
            String username = props.getProperty(Environment.USER);
            String password = props.getProperty(Environment.PASS);
            dbcpProperties.put("username", username);
            dbcpProperties.put("password", password);

            // Isolation level
            String isolationLevel = props.getProperty(Environment.ISOLATION);
            if ((isolationLevel != null) && (isolationLevel.trim().length() > 0)) {
                dbcpProperties.put("defaultTransactionIsolation", isolationLevel);
            }

            // Turn off autocommit (unless autocommit property is set)
            String autocommit = props.getProperty(AUTOCOMMIT);
            if ((autocommit != null) && (autocommit.trim().length() > 0)) {
                dbcpProperties.put("defaultAutoCommit", autocommit);
            } else {
                dbcpProperties.put("defaultAutoCommit", String.valueOf(Boolean.FALSE));
            }

            // Pool size
```

```
        String poolSize = props.getProperty(Environment.POOL_SIZE);
        if ((poolSize != null) && (poolSize.trim().length() > 0)
            && (Integer.parseInt(poolSize) > 0))  {
            dbcpProperties.put("maxActive", poolSize);
        }

        // Copy all "driver" properties into "connectionProperties"
        Properties driverProps = ConnectionProviderFactory.getConnectionProperties(props);
        if (driverProps.size() > 0) {
            StringBuffer connectionProperties = new StringBuffer();
            for (Iterator iter = driverProps.entrySet().iterator(); iter.hasNext();) {
                Map.Entry entry = (Map.Entry) iter.next();
                String key = (String) entry.getKey();
                String value = (String) entry.getValue();
                connectionProperties.append(key).append('=').append(value);
                if (iter.hasNext()) {
                    connectionProperties.append(';');
                }
            }
            dbcpProperties.put("connectionProperties", connectionProperties.toString());
        }

        // Copy all DBCP properties removing the prefix
        for (Iterator iter = props.entrySet().iterator() ; iter.hasNext() ;) {
            Map.Entry entry = (Map.Entry) iter.next();
            String key = (String) entry.getKey();
            if (key.startsWith(PREFIX)) {
                String property = key.substring(PREFIX.length());
                String value = (String) entry.getValue();
                dbcpProperties.put(property, value);
            }
        }

        // Backward-compatibility
        if (props.getProperty(DBCP_PS_MAXACTIVE) != null) {
            dbcpProperties.put("poolPreparedStatements", String.valueOf(Boolean.TRUE));
            dbcpProperties.put("maxOpenPreparedStatements", props.getProperty(DBCP_PS_MAXACTIVE));
        }

        // Some debug info
        if (log.isDebugEnabled()) {
            log.debug("Creating a DBCP BasicDataSource with the following DBCP factory properties:");
            StringWriter sw = new StringWriter();
            dbcpProperties.list(new PrintWriter(sw, true));
            log.debug(sw.toString());
        }

        // Let the factory create the pool
        ds = (BasicDataSource) BasicDataSourceFactory.createDataSource(dbcpProperties);

        // The BasicDataSource has lazy initialization
        // borrowing a connection will start the DataSource
        // and make sure it is configured correctly.
        Connection conn = ds.getConnection();
        conn.close();

        // Log pool statistics before continuing.
        logStatistics();
    }
    catch (Exception e) {
        String message = "Could not create a DBCP pool";
        log.fatal(message, e);
        if (ds != null) {
            try {
                ds.close();
            }
            catch (Exception e2) {
                // ignore
            }
            ds = null;
        }
```

```
            throw new HibernateException(message, e);
        }
        log.debug("Configure DBCPConnectionProvider complete");
    }

    public Connection getConnection() throws SQLException {
        Connection conn = null;
        try {
                conn = ds.getConnection();
        }
        finally {
            logStatistics();
        }
        return conn;
    }

    public void closeConnection(Connection conn) throws SQLException {
        try {
            conn.close();
        }
        finally {
            logStatistics();
        }
    }

    public void close() throws HibernateException {
        log.debug("Close DBCPConnectionProvider");
        logStatistics();
        try {
            if (ds != null) {
                ds.close();
                    ds = null;
            }
            else {
                log.warn("Cannot close DBCP pool (not initialized)");
            }
        }
        catch (Exception e) {
            throw new HibernateException("Could not close DBCP pool", e);
        }
        log.debug("Close DBCPConnectionProvider complete");
    }

    protected void logStatistics() {
        if (log.isInfoEnabled()) {
            log.info("active: " + ds.getNumActive() + " (max: " + ds.getMaxActive() + ")   "
                    + "idle: " + ds.getNumIdle() + "(max: " + ds.getMaxIdle() + ")");
        }
    }

    public boolean supportsAggressiveRelease()
    {
        return false;
    }
}
```