◆ ◆ ◆ CHAPTER 14

Using the JDBC API for Database Access

This chapter describes how to use the Java Database Connectivity (JDBC) API for database access with the Oracle GlassFish Server. This chapter also provides high level JDBC implementation instructions for servlets and EJB components using the GlassFish Server. If the JDK version 1.6 is used, the GlassFish Server supports the JDBC 4.0 API.

The JDBC specifications are available at `http://java.sun.com/products/jdbc/download.html`.

A useful JDBC tutorial is located at `http://java.sun.com/docs/books/tutorial/jdbc/index.html`.

Note – The GlassFish Server does not support connection pooling or transactions for an application's database access if it does not use standard Java EE `DataSource` objects.

This chapter discusses the following topics:

Statements

The following features pertain to statements:

Using an Initialization Statement

You can specify a statement that executes each time a physical connection to the database is created (not reused) from a JDBC connection pool. This is useful for setting request or session specific properties and is suited for homogeneous requests in a single application. Set the Init SQL attribute of the JDBC connection pool to the SQL string to be executed in one of the following ways:

- Enter an Init SQL value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--initsql` option in the `asadmin create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.
- Specify the `init-sql` option in the `asadmin set` command. For example:

  ```
  asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.init-sql="sql-string"
  ```

  For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Setting a Statement Timeout

An abnormally long running JDBC query executed by an application may leave it in a hanging state unless a timeout is explicitly set on the statement. Setting a statement timeout guarantees that all queries automatically time out if not completed within the specified period. When statements are created, the `queryTimeout` is set according to the statement timeout setting. This works only when the underlying JDBC driver supports `queryTimeout` for `Statement`, `PreparedStatement`, `CallableStatement`, and `ResultSet`.

You can specify a statement timeout in the following ways:

- Enter a Statement Timeout value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.

- Specify the `--statementtimeout` option in the `asadmin create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Statement Leak Detection and Leaked Statement Reclamation

If statements are not closed by an application after use, it is possible for the application to run out of cursors. Enabling statement leak detection causes statements to be considered as leaked if they are not closed within a specified period. Additionally, leaked statements can reclaimed automatically.

To enable statement leak detection, set Statement Leak Timeout In Seconds for the JDBC connection pool to a positive, nonzero value in one of the following ways:

- Specify the `--statementleaktimeout` option in the `create-jdbc-connection-pool` subcommand. For more information, see `create-jdbc-connection-pool`(1).
- Specify the `statement-leak-timeout-in-seconds` option in the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.statement-leak-timeout-in-seconds=300
```

When selecting a value for Statement Leak Timeout In Seconds, make sure that:

- It is less than the Connection Leak Timeout; otherwise, the connection could be closed before the statement leak is recognized.
- It is greater than the Statement Timeout; otherwise, a long running query could be mistaken as a statement leak.

After enabling statement leak detection, enable leaked statement reclamation by setting Reclaim Leaked Statements for the JDBC connection pool to a `true` value in one of the following ways:

- Specify the `--statementleakreclaim=true` option in the `create-jdbc-connection-pool` subcommand. For more information, see `create-jdbc-connection-pool`(1).
- Specify the `statement-leak-reclaim` option in the `set` subcommand. For example:

  ```
  asadmin set resources.jdbc-connection-pool.pool-name.statement-leak-reclaim=true
  ```

Statement Caching

Statement caching stores statements, prepared statements, and callable statements that are executed repeatedly by applications in a cache, thereby improving performance. Instead of the statement being prepared each time, the cache is searched for a match. The overhead of parsing and creating new statements each time is eliminated.

Statement caching is usually a feature of the JDBC driver. The GlassFish Server provides caching for drivers that do not support caching. To enable this feature, set the Statement Cache Size for the JDBC connection pool in one of the following ways:

- Enter a Statement Cache Size value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--statementcachesize` option in the `asadmin create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.
- Specify the `statement-cache-size` option in the `asadmin set` command. For example:

  ```
  asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.statement-cache-size=10
  ```

  For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

By default, this attribute is set to zero and the statement caching is turned off. To enable statement caching, you can set any positive nonzero value. The built-in cache eviction strategy is LRU-based (Least Recently Used). When a connection pool is flushed, the connections in the statement cache are recreated.

Statement Tracing

You can trace the SQL statements executed by applications that use a JDBC connection pool. Set the SQL Trace Listeners attribute to a comma-separated list of trace listener implementation classes in one of the following ways:

- Enter an SQL Trace Listeners value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--sqltracelisteners` option in the `asadmin create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.
- Specify the `sql-trace-listeners` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.sql-trace-listeners=listeners
```

  For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

The GlassFish Server provides a public interface, org.glassfish.api.jdbc.SQLTraceListener, that implements a means of recording `SQLTraceRecord` objects. To make custom implementations of this interface available to the GlassFish Server, place the implementation classes in *as-install*`/lib`.

The GlassFish Server provides an SQL tracing logger to log the SQL operations in the form of `SQLTraceRecord` objects in the `server.log` file. The module name under which the SQL

operation is logged is `javax.enterprise.resource.sqltrace`. SQL traces are logged as FINE messages along with the module name to enable easy filtering of the SQL logs. A sample SQL trace record looks like this:

```
[#|2009-11-27T15:46:52.202+0530|FINE|glassfishv3.0|javax.enterprise.resource.sqltrace.com.sun.gjc.util
|_ThreadID=29;_ThreadName=Thread-1;ClassName=com.sun.gjc.util.SQLTraceLogger;MethodName=sqlTrace;
|ThreadID=77 | ThreadName=p: thread-pool-1; w: 6 | TimeStamp=1259317012202
| ClassName=com.sun.gjc.spi.jdbc40.PreparedStatementWrapper40 | MethodName=executeUpdate
| arg[0]=insert into table1(colName) values(100) | arg[1]=columnNames | |#]
```

This trace shows that an `executeUpdate(String sql, String columnNames)` operation is being done.

When SQL statement tracing is enabled and JDBC connection pool monitoring is enabled, GlassFish Server maintains a tracing cache of recent queries and their frequency of use. The following JDBC connection pool properties can be configured to control this cache and the monitoring statistics available from it:

`time-to-keep-queries-in-minutes`
: Specifies how long in minutes to keep a query in the tracing cache, tracking its frequency of use. The default value is 5 minutes.

`number-of-top-queries-to-report`
: Specifies how many of the most used queries, in frequency order, are listed the monitoring report. The default value is 10 queries.

Set these parameters in one of the following ways:

- Add them as properties in the Edit JDBC Connection Pool Properties page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify them using the `--property` option in the `create-jdbc-connection-pool` subcommand. For more information, see `create-jdbc-connection-pool`(1).
- Set them using the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.property.time-to-keep-queries-in-minutes=10
```

Connections

The following features pertain to connections:

- "Handling Invalid Connections" on page 265

Transparent Pool Reconfiguration

When the properties or attributes of a JDBC connection pool are changed, the connection pool is destroyed and re-created. Normally, applications using the connection pool must be redeployed as a consequence. This restriction can be avoided by enabling transparent JDBC connection pool reconfiguration. When this feature is enabled, applications do not need to be redeployed. Instead, requests for new connections are blocked until the reconfiguration operation completes. Connection requests from any in-flight transactions are served using the old pool configuration so as to complete the transaction. Then, connections are created using the pool's new configuration, and any blocked connection requests are served with connections from the re-created pool..

To enable transparent JDBC connection pool reconfiguration, set the `dynamic-reconfiguration-wait-timeout-in-seconds` property of the JDBC connection pool to a positive, nonzero value in one of the following ways:

- Add it as a property in the Edit JDBC Connection Pool Properties page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify it using the `--property` option in the `create-jdbc-connection-pool` subcommand. For more information, see `create-jdbc-connection-pool`(1).
- Set it using the `set` subcommand. For example:

```
asadmin set resources.jdbc-connection-pool.pool-name.property.dynamic-reconfiguration-wait-timeout-in-seconds=15
```

This property specifies the time in seconds to wait for in-use connections to close and in-flight transactions to complete. Any connections in use or transaction in flight past this time must be retried.

Disabling Pooling

To disable connection pooling, set the Pooling attribute to false. The default is true. You can enable or disable connection pooling in one of the following ways:

- Enter a Pooling value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--pooling` option in the `asadmin create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.
- Specify the `pooling` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.pooling=false
```

For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

The `pooling` option and the system property `com.sun.enterprise.connectors.SwitchoffACCConnectionPooling`, which turns off connection pooling in the Application Client Container, do not affect each other.

An exception is thrown if `associate-with-thread` is set to `true` and pooling is disabled. An exception is thrown if you attempt to flush a connection pool when pooling is disabled. A warning is logged if the following attributes are used, because they are useful only in a pooled environment:

- `connection-validation`
- `validate-atmost-once-period`
- `match-connections`
- `max-connection-usage`
- `idle-timeout`

Associating Connections with Threads

To associate connections with a thread, set the Associate With Thread attribute to `true`. The default is `false`. A `true` setting allows connections to be saved as `ThreadLocal` in the calling thread. Connections get reclaimed only when the calling thread dies or when the calling thread is not in use and the pool has run out of connections. If the setting is `false`, the thread must obtain a connection from the pool each time the thread requires a connection.

The Associate With Thread attribute associates connections with a thread such that when the same thread is in need of connections, it can reuse the connections already associated with that thread. In this case, the overhead of getting connections from the pool is avoided. However, when this value is set to `true`, you should verify that the value of the Max Pool Size attribute is comparable to the Max Thread Pool Size attribute of the thread pool. If the Max Thread Pool Size value is much higher than the Max Pool Size value, a lot of time is spent associating connections with a new thread after dissociating them from an older one. Use this attribute in cases where the thread pool should reuse connections to avoid this overhead.

You can set the Associate With Thread attribute in the following ways:

- Enter an Associate With Thread value in the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--associatewiththread` option in the `asadmin create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.
- Specify the `associate-with-thread` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.associate-with-thread=true
```

For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Custom Connection Validation

You can specify a custom implementation for Connection Validation that is faster or optimized for a specific database. Set the Validation Method attribute to the value `custom-validation`. (Other validation methods available are `table` (the default), `auto-commit`, and `meta-data`.) The GlassFish Server provides a public interface, org.glassfish.api.jdbc.ConnectionValidation, which you can implement to plug in your implementation. A new attribute, Validation Classname, specifies the fully qualified name of the class that implements the ConnectionValidation interface. The Validation Classname attribute is required if Connection Validation is enabled and Validation Method is set to Custom Validation.

To enable this feature, set Connection Validation, Validation Method, and Validation Classname for the JDBC connection pool in one of the following ways:

- Enter Connection Validation, Validation Method, and Validation Classname values in the Edit Connection Pool Advanced Attributes page in the Administration Console. You can select from among validation class names for common databases in the Validation Classname field. For more information, click the Help button in the Administration Console.
- Specify the `--isconnectionvalidatereq`, `--validationmethod`, and `--validationclassname` options in the `asadmin create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.
- Specify the `is-connection-validation-required`, `connection-validation-method`, and `validation-classname` options in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.MyPool.is-connection-validation-required=true
asadmin set domain1.resources.jdbc-connection-pool.MyPool.connection-validation-method=custom-validation
asadmin set domain1.resources.jdbc-connection-pool.MyPool.validation-classname=impl-class
```

For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

By default, optimized validation mechanisms are provided for DB2, Java DB, MSSQL, MySQL, Oracle, PostgreSQL and Sybase databases. Additionally, for JDBC 4.0 compliant database drivers, a validation mechanism is provided that uses the `Connection.isValid(0)` implementation.

Sharing Connections

When multiple connections acquired by an application use the same JDBC resource, the connection pool provides connection sharing within the same transaction scope. For example, suppose Bean A starts a transaction and obtains a connection, then calls a method in Bean B. If

Bean B acquires a connection to the same JDBC resource with the same sign-on information, and if Bean A completes the transaction, the connection can be shared.

Connections obtained through a resource are shared only if the resource reference declared by the Java EE component allows it to be shareable. This is specified in a component's deployment descriptor by setting the `res-sharing-scope` element to `Shareable` for the particular resource reference. To turn off connection sharing, set `res-sharing-scope` to `Unshareable`.

For general information about connections and JDBC URLs, see Chapter 11, "Administering Database Connectivity," in *GlassFish Server Open Source Edition 3.1 Administration Guide*.

Marking Bad Connections

The `DataSource` implementation in the GlassFish Server provides a `markConnectionAsBad` method. A marked bad connection is removed from its connection pool when it is closed. The method signature is as follows:

```
public void markConnectionAsBad(java.sql.Connection con)
```

For example:

```
com.sun.appserv.jdbc.DataSource ds=
   (com.sun.appserv.jdbc.DataSource)context.lookup("dataSource");
Connection con = ds.getConnection();
Statement stmt = null;
try{
   stmt = con.createStatement();
   stmt.executeUpdate("Update");
}
catch (BadConnectionException e){
   ds.markConnectionAsBad(con) //marking it as bad for removal
}
finally{
   stmt.close();
   con.close(); //Connection will be destroyed during close.
}
```

Handling Invalid Connections

If a `ConnectionErrorOccured` event occurs, the GlassFish Server considers the connection invalid and removes the connection from the connection pool. Typically, a JDBC driver generates a `ConnectionErrorOccured` event when it finds a `ManagedConnection` object unusable. Reasons can be database failure, network failure with the database, fatal problems with the connection pool, and so on.

If the `fail-all-connections` setting in the connection pool configuration is set to `true`, and a single connection fails, all connections are closed and recreated. If this setting is `false`, individual connections are recreated only when they are used. The default is `false`.

The `is-connection-validation-required` setting specifies whether connections have to be validated before being given to the application. If a resource's validation fails, it is destroyed, and a new resource is created and returned. The default is `false`.

The `prefer-validate-over-recreate` property specifies that validating idle connections is preferable to closing them. This property has no effect on non-idle connections. If set to `true`, idle connections are validated during pool resizing, and only those found to be invalid are destroyed and recreated. If `false`, all idle connections are destroyed and recreated during pool resizing. The default is `false`.

You can set the `fail-all-connections`, `is-connection-validation-required`, and `prefer-validate-over-recreate` configuration settings during creation of a JDBC connection pool. Or, you can use the `asadmin set` command to dynamically reconfigure a setting. For example:

```
asadmin set server.resources.jdbc-connection-pool.JCPool1.fail-all-connections="true"
asadmin set server.resources.jdbc-connection-pool.JCPool1.is-connection-validation-required="true"
asadmin set server.resources.jdbc-connection-pool.JCPool1.property.prefer-validate-over-recreate="true"
```

For details, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

The interface ValidatingManagedConnectionFactory exposes the method `getInvalidConnections` to allow retrieval of the invalid connections. The GlassFish Server checks if the JDBC driver implements this interface, and if it does, invalid connections are removed when the connection pool is resized.

Connection Wrapping

The following features pertain to connection wrapping:

Wrapping Connections

If the Wrap JDBC Objects option is `true` (the default), wrapped JDBC objects are returned for `Statement`, `PreparedStatement`, `CallableStatement`, `ResultSet`, and `DatabaseMetaData`.

This option ensures that `Statement.getConnection()` is the same as `DataSource.getConnection()`. Therefore, this option should be `true` when both `Statement.getConnection()` and `DataSource.getConnection()` are done.

You can specify the Wrap JDBC Objects option in the following ways:

- Check or uncheck the Wrap JDBC Objects box on the Edit Connection Pool Advanced Attributes page in the Administration Console. For more information, click the Help button in the Administration Console.
- Specify the `--wrapjdbcobjects` option in the `asadmin` `create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Obtaining a Physical Connection From a Wrapped Connection

The `DataSource` implementation in the GlassFish Server provides a `getConnection` method that retrieves the JDBC driver's `SQLConnection` from the GlassFish Server's `Connection` wrapper. The method signature is as follows:

```
public java.sql.Connection getConnection(java.sql.Connection con)
throws java.sql.SQLException
```

For example:

```
InitialContext ctx = new InitialContext();
com.sun.appserv.jdbc.DataSource ds = (com.sun.appserv.jdbc.DataSource)
   ctx.lookup("jdbc/MyBase");
Connection con = ds.getConnection();
Connection drivercon = ds.getConnection(con); //get physical connection from wrapper
// Do db operations.
// Do not close driver connection.
con.close(); // return wrapped connection to pool.
```

Using the `Connection.unwrap()` Method

If the JDK version 1.6 is used, the GlassFish Server supports JDBC 4.0 if the JDBC driver is JDBC 4.0 compliant. Using the `Connection.unwrap()` method on a vendor-provided interface returns an object or a wrapper object implementing the vendor-provided interface, which the application can make use of to do vendor-specific database operations. Use the `Connection.isWrapperFor()` method on a vendor-provided interface to check whether the connection can provide an implementation of the vendor-provided interface. Check the JDBC driver vendor's documentation for information on these interfaces.

Allowing Non-Component Callers

You can allow non-Java-EE components, such as servlet filters, lifecycle modules, and third party persistence managers, to use this JDBC connection pool. The returned connection is automatically enlisted with the transaction context obtained from the transaction manager. Standard Java EE components can also use such pools. Connections obtained by non-component callers are not automatically closed at the end of a transaction by the container. They must be explicitly closed by the caller.

You can enable non-component callers in the following ways:

- Check the Allow Non Component Callers box on the Edit Connection Pool Advanced Attributes page in the Administration Console. The default is `false`. For more information, click the Help button in the Administration Console.
- Specify the `--allownoncomponentcallers` option in the `asadmin create-jdbc-connection-pool` command. For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.
- Specify the `allow-non-component-callers` option in the `asadmin set` command. For example:

```
asadmin set domain1.resources.jdbc-connection-pool.DerbyPool.allow-non-component-callers=true
```

  For more information, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.
- Create a JDBC resource with a `__pm` suffix.

Accessing a `DataSource` using the `Synchronization.beforeCompletion()` method requires setting Allow Non Component Callers to `true`. For more information about the Transaction Synchronization Registry, see "The Transaction Manager, the Transaction Synchronization Registry, and `UserTransaction`" on page 277.

Using Application-Scoped Resources

You can define an application-scoped database or other resource for an enterprise application, web module, EJB module, connector module, or application client module by supplying a `glassfish-resources.xml` deployment descriptor file. For details, see "Application-Scoped Resources" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

Restrictions and Optimizations

This section discusses restrictions and performance optimizations that affect using the JDBC API.

Disabling Stored Procedure Creation on Sybase

By default, DataDirect and Oracle JDBC drivers for Sybase databases create a stored procedure for each parameterized `PreparedStatement`. On the GlassFish Server, exceptions are thrown when primary key identity generation is attempted. To disable the creation of these stored procedures, set the property `PrepareMethod=direct` for the JDBC connection pool.