Apache Derby

The Apache DB Project
http://db.apache.org/

Last Published: 05/07/2020 21:39:00

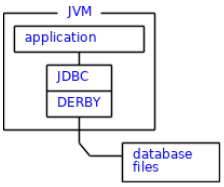
Font size: Reset -a +a

Step 3: Embedded Derby

Introduction

When an application accesses a Derby database using the Embedded Derby JDBC driver, the Derby engine does not run in a separate process, and there are no separate database processes to start up and shut down. Instead, the Derby database engine runs inside the same Java Virtual Machine (JVM) as the application. So, Derby becomes part of the application just like any other jar file that the application uses. Figure 1 depicts this embedded architecture.

Figure 1: Derby Embedded Architecture



This section shows how to compile and run a simple Java application using the Derby Embedded JDBC driver. The information presented is deliberately simple. For complete information, see the Derby Developer's Guide.

Set the environment

To set up the environment, follow the "Configure Embedded Derby" instructions.

Sample Application

Copy sample application

The Derby software includes a sample application called `SimpleApp.java`. In Derby 10.2 and above it is located in the `DERBY_INSTALL/demo/programs/simple/` directory. (In Derby 10.1 it is located in the `DERBY_INSTALL/demo/simple/` directory.) Copy that application into your current directory.

By default this application runs in embedded mode and does the following:

- Starts up the Derby engine
- Creates and connects to a database
- Creates a table
- Inserts data
- Updates data
- Selects data
- Drops a table
- Disconnects
- Shuts down Derby

A quick look at the code

The `SimpleApp.java` application spends most of its time creating a table, inserting data into that table, and selecting the data back, demonstrating many JDBC calls as it does so. This section highlights the JDBC calls that make this specifically an embedded Derby application. The "Derby Network Server" section shows how the same JDBC calls turn the same code into a client/server application.

Load the Embedded JDBC Driver

The `SimpleApp` application loads the Derby Embedded JDBC driver and starts Derby up with this code:

```
public String driver = "org.apache.derby.jdbc.EmbeddedDriver";
...
Class.forName(driver).newInstance();
```

Get an Embedded Connection

The `SimpleApp` application creates and connects to the `derbyDB` database with this code:

```
public String protocol = "jdbc:derby:";
...
conn = DriverManager.getConnection(protocol + "derbyDB;create=true", props);
```

That embedded connection URL, fully constructed, looks like this:

```
jdbc:derby:derbyDB;create=true
```

Shut Derby down

A clean shutdown performs a checkpoint and releases resources. If an embedded application doesn't shut down Derby, a checkpoint won't be performed. Nothing bad will happen; it just means that the next connection will be slower because Derby will run its recovery code.

Code to shut down a specific database looks like this:

```
DriverManager.getConnection("jdbc:derby:MyDbTest;shutdown=true");
```

Code to shut down all databases and the Derby engine looks like this:

```
DriverManager.getConnection("jdbc:derby:;shutdown=true");
```

The `SimpleApp.java` code uses the second call to shut down all databases and the engine. You might also notice that it tests for a SQL exception. A clean shutdown always throws SQL exception `XJ015`, which can be ignored.

Compile sample application

Compile the sample application as shown below:

```
javac SimpleApp.java
```

You can safely ignore any compile warnings that might pop up.

Run sample application

Run the sample application like this:

```
java SimpleApp
```

You should see the output shown below:

```
SimpleApp starting in embedded mode.
Loaded the appropriate driver.
Connected to and created database derbyDB
Created table derbyDB
Inserted 1956 Webster
Inserted 1910 Union
Updated 1956 Webster to 180 Grand
Updated 180 Grand to 300 Lakeshore
Verified the rows
Dropped table derbyDB
Closed result set and statement
Committed transaction and closed connection
Database shut down normally
SimpleApp finished
```

If instead you see an error like the one below, it means the class path is not correctly set:

```
java SimpleApp
SimpleApp starting in embedded mode.
exception thrown:
java.lang.ClassNotFoundException: org.apache.derby.jdbc.EmbeddedDriver
        at java.net.URLClassLoader$1.run(URLClassLoader.java:199)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:187)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:289)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:274)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:235)
        at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:302)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:141)
        at SimpleApp.go(SimpleApp.java:77)
        at SimpleApp.main(SimpleApp.java:61)
SimpleApp finished
```

For help resolving class path problems, see the "Configure Embedded Derby" and "Verify Derby" instructions.

Embedded Derby supports multiple users in one JVM

Derby supports multiple connections to a given database. An example of an embedded application that manages concurrent users is a Tomcat or Geronimo application server that embeds Derby. Any number of users can execute Web applications that access a database through that Web server.

However, only one JVM may boot ("open") that database, so multiple applications running in *different* JVMs cannot access the same database. To understand this better, do the following.

The `SimpleApp` application created a database called `derbyDB`. In one window connect to this database with `ij` as shown below:

```
java org.apache.derby.tools.ij
ij version 10.4
ij> connect 'jdbc:derby:derbyDB';
ij>
```

Now, in another window try to run your `SimpleApp` application:

```
java SimpleApp
SimpleApp starting in embedded mode
Loaded the appropriate driver

----- SQLException -----
  SQL State:  XJ040
  Error Code: 40000
```

```
  Message:    Failed to start database 'derbyDB', see the next exception for details.

----- SQLException -----
  SQL State:  XSDB6
  Error Code: 45000
  Message:    Another instance of Derby may have already booted the database C:\Apache\db-derby-10.4.1.3-bin\tutorial\derbyDB.
SimpleApp finished
```

The connection that `SimpleApp` tries to establish fails because another application, `ij`, is already connected to the `derbyDB` database.

The problem is `ij` and `SimpleApp` are running in different JVMs and a given database can only be accessed from one JVM. The first application to connect "wins", in this case `ij`, and prevents other applications from connecting.

In fact, you *can* establish multiple connections to a given database from different JVMs. That's the topic of the next section, "Step 4: Derby Network Server".

Last Published: 05/07/2020 21:39:00