JDBC Driver Manager

3.0 Driver Manager

The DriverManager class is working between the user and the drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. JDBC requires a Driver class to register itself by calling the method DriverManager.registerDriver when it is instantiated. The act of instantiating a Driver class thus enters it in the DriverManager's list.

3.1 Keeping Track of Available Drivers

All Driver classes should be written with a static section (a static initializer) that creates an instance of the class and then registers it with the DriverManager class when it is loaded. A Driver class is loaded, and therefore automatically registered with the DriverManager, in one of two ways:

3.1.1 By calling the method "Class.forName()".

This explicitly loads the driver class. This creates the new instances of the driver and also calls the DriverManager.registerDriver with that instance as the parameter. Now this driver is in the list of DriverManager's list of drivers and available for creating a connection. This way is the recommended one for using the DriverManager. The following code loads the class "oracle.jdbc.driver.OracleDriver"

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

3.1.2 By adding the Driver class to the java.lang.System property "jdbc.drivers".

You can also mention your driver name as a value for this key jdbc.drivers in java.lang.System property. When the DriverManager class is initialized, it looks first for the system property "jdbc.drivers" and automatically load these drivers classes into it.

In both of these cases, it is the responsibility of the newly loaded Driver class to register itself by calling DriverManager.registerDriver. As mentioned, this should be done automatically when the class is loaded. For security reasons, the JDBC management layer will keep track of which class loader provided which driver. Then, when the DriverManager class is opening a connection, it will use only drivers that come from the local file system or from the same class loader as the code issuing the request for a connection.

3.1.3 Note : JDBC 4.0 updates

The DriverManager methods getConnection and getDrivers have been enhanced to support the Java Standard Edition Service Provider mechanism. JDBC 4.0 Drivers must include the file META-INF/services/java.sql.Driver. This file contains the name of the JDBC drivers implementation of java.sql.Driver.

For example, to load the my.sql.Driver class, the META-INF/services/java.sql.Driver file would contain the entry: my.sql.Driver. Applications no longer need to explictly load JDBC drivers using Class.forName(). Existing programs which currently load JDBC drivers using Class.forName() will continue to work without modification.

3.2 Establishing a Connection

The lists of driver classes in the DriverManager are available for establishing a connection with database.When a request made to call the DriverManager.getConnection method, DriverManger tries to use each driver in the order it was registered.

First preference goes to the drivers mentioned in jdbc.drivers property. If more than one JDBC drivers is capable of connecting to a given URL, DriverManger will use the first driver it finds that can successfully connect to the given URL. Internally, DriverManager tests the driver by calling the method Driver.connect on each one by passing the Connection URL.

3.2.1 Note.

With the addition DataSource object from JDBC 2.0 to establish a connection with database, it is recommended to use Datasource object. It has several advantages over the DriverManager.

3.3 References

http://download.oracle.com/javase/1,5,0/docs/api/java/sql/DriverManager.html

http://www.herongyang.com/JDBC/Overview-JDBC-DriverManager.html

Explains how driver is loading in webapp - https://issues.apache.org/bugzilla/show_bug.cgi?id=48214

http://www.netcluesoft.com/jdbc-drivers-and-classforname.html