Connecting to DB through Datasource

`JDBC 2.0` optional package

- **`javax.sql package`**
- Includes the following:
  - **`DataSource`** interface
  - Connection Pooling
  - Distributed Transactions
  - **`RowSet`**

DataSource

- **`DataSource`** interface is used as an alternative to **`DriverManager`** of JDBC 2.0 core package.
- It makes the application more *portable* and makes the code *easier to maintain*.
- A **`DataSource`** object refers to the real world data source (relational database or spreadsheet etc.)
- **`DataSource`** object is created (typically by an application server) when the details like the URL of the data source and the Driver name is provided.

JNDI and DataSources

- **`DataSource`** object thus created is then registered with a JNDI naming service (which maps the **`DataSource`** object with a name).
- Once this is done, any application can retrieve the **`DataSource`** object from the JNDI naming service by providing the name.
- Once the **`DataSource`** is obtained, the **`Connection`** object can be created after which the code will work with JDBC 2.0 core package as usual.

Steps to get Connection object from `DataSource`

Step 1: Get the **`DataSource`** object

```
Context ctx=new InitialContext();
DataSource ds= (DataSource )
 ctx.lookup(“jdbc/MyDSName”);
```

Step 2: Get the connection object

```
Connection con=
 ds.getConnection(“username”,”passwor
 d”);
```

Advantages

- **Portability**: Since, information such as the URL of the data source location and Driver Class Name are not hard-coded in the application, it makes the application independent of database (or data source) vendor.
- **Easy Maintenance**: If the url of data source changes, it just involves making one change in data source configuration. Application need not be touched at all!
- **Connection Pooling:** Connection Pooling is a mechanism whereby a collection of connection objects is maintained by the application.

J2EE Application Server’s support for `DataSource`

- J2EE Application Server comes with some JNDI naming and directory services (LDAP Server etc.).
- Most Application Servers provide a tool that allows application developers or administrators to configure **`DataSource`**s to this JNDI service.
- Advantage with this is that the application developer can defer the decision of which database to use until the deployment time!

Distributed Transactions

- From an application developer's point of view there is no difference between Connection obtained by the **`DataSource`** object and the **`DriverManager`** except in case of transactions.
- With **`Connection`** object obtained from **`DataSource`**, distributed transaction is automatically supported.
- Therefore the application using **`Connection`** object obtained from **`DataSource`** cannot call commit or rollback methods directly.

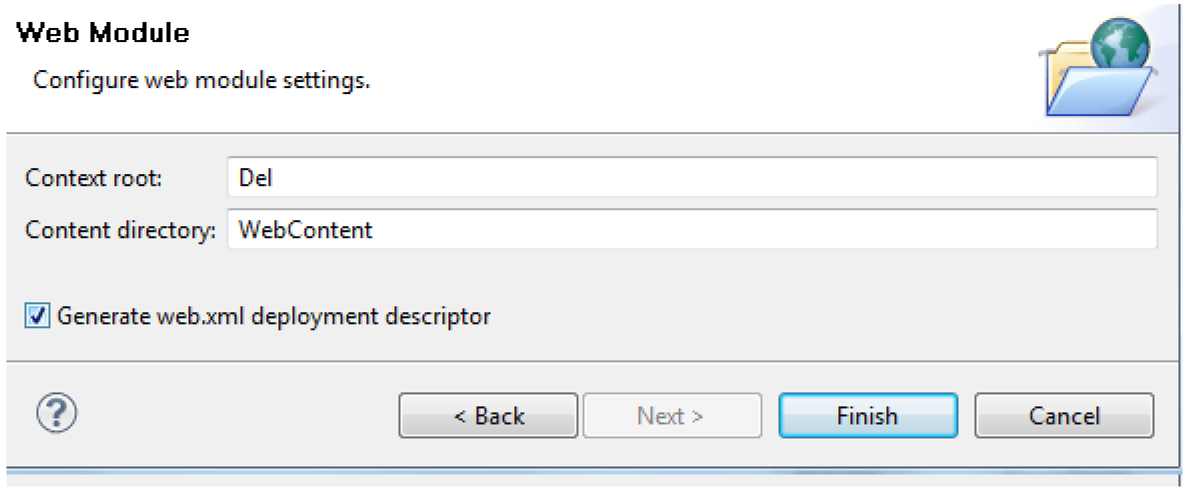
Activity

Steps to work with DataSource in Tomcat7.0

1. Create a web application with web.xml file.
2. Create a configuration file for DataSource
3. Add <resource-ref> in web.xml
4. Add libraries
5. Write code to connect to the database

Create a web application with web.xml file

- The steps to create a new web application with web.xml file is same as that of creating a new Dynamic Web Project, except that when creating, continue to click on the Next buttons until you find a page where a checkbox “Generate web.xml deployment descriptor” . Tick this checkbox and Finish.



Web Module

Configure web module settings.

Context root: Del

Content directory: WebContent

[✓] Generate web.xml deployment descriptor

Configuring DataSource in Tomcat

- 2 ways to configure DataSource
  1. Add **`context.xml`** in the individual web application inside META-INF folder.
     - **Recommended way**
     - When running from Eclipse WTP extra configuration needs to be done on the server configuration file.

     OR
  2. add **`<Context>`** in the **`server.xml file`** in **`Tomcat conf`** folder.
     - Used when database is commonly accessed my multiple applications

1. context.xml in individual application

Create a context.xml file inside META-INF of your web application.

```
<?xml version="1.0" encoding="UTF-8"?>
    <Context docBase="/SRASYSWEB"
        path="SRASYSWEB"
reloadable="true">
        <Resource name="jdbc/Test"
            auth="Container"
            type="javax.sql.DataSource"
            maxActive="100"
            maxIdle="30"
            maxWait="-1"
            username="root"
            password="root"
      driverClassName="oracle.jdbc.driver.OracleDriver"
            url=" jdbc:oracle:thin:@127.0.0.1:1521:xe">
        </Resource>
    </Context>
```

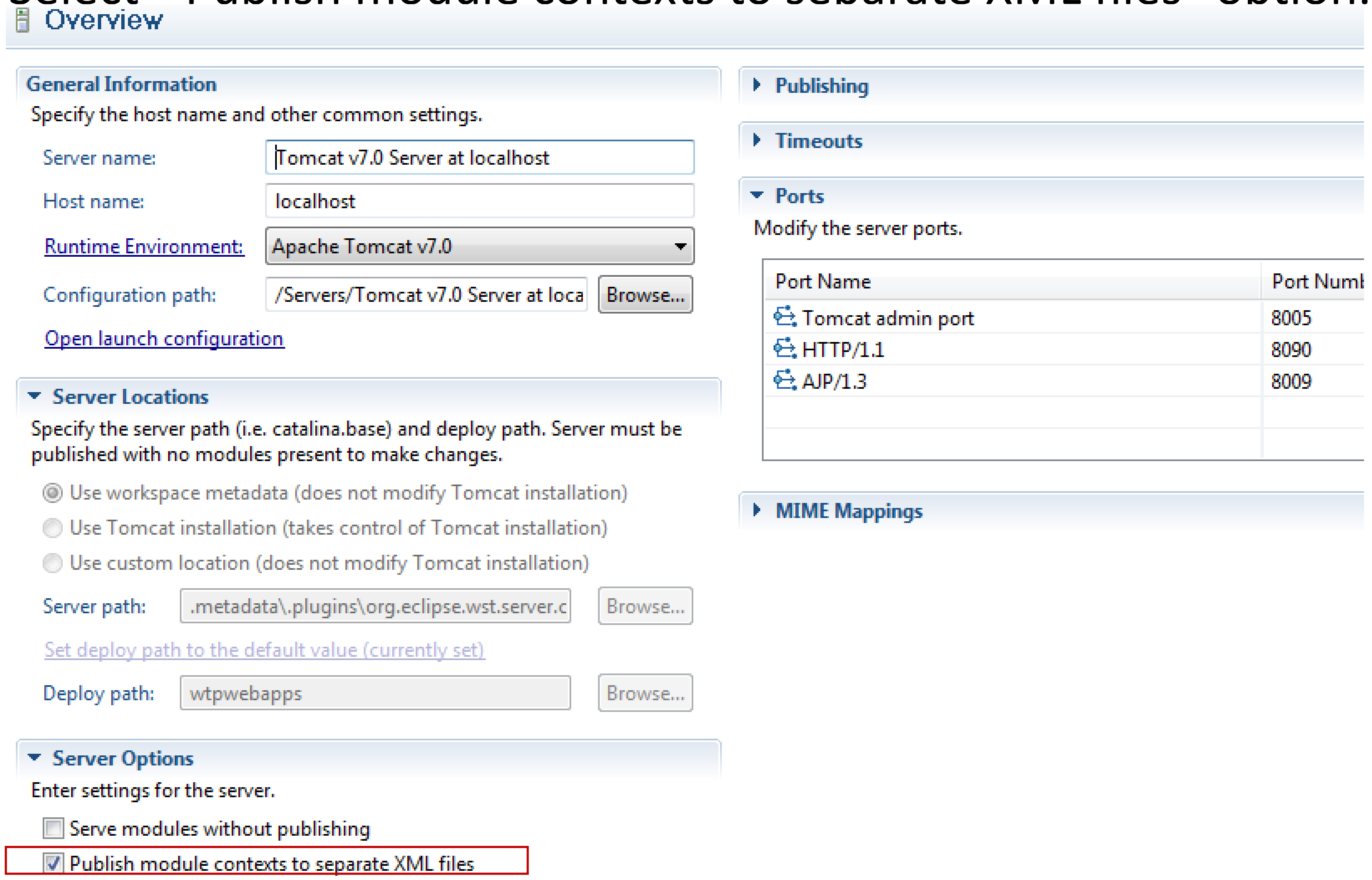
web application context

JNDI name

Configuration with respect to Oracle

Configuring server file in Eclipse WTP

- Double click on the Tomcat v 7.0 Server in the server panel .
- Select "Publish module contexts to separate XML files" option.



Overview

General Information

Specify the host name and other common settings.

Server name:	Tomcat v7.0 Server at localhost
Host name:	localhost
Runtime Environment:	Apache Tomcat v7.0
Configuration path:	/Servers/Tomcat v7.0 Server at loca

Open launch configuration

Server Locations

Specify the server path (i.e. catalina.base) and deploy path. Server must be published with no modules present to make changes.

- ● Use workspace metadata (does not modify Tomcat installation)
- ○ Use Tomcat installation (takes control of Tomcat installation)
- ○ Use custom location (does not modify Tomcat installation)

Server path: .metadata\.plugins\org.eclipse.wst.server.c

Set deploy path to the default value (currently set)

Deploy path: wtpwebapps

Server Options

Enter settings for the server.

- [ ] Serve modules without publishing
- [✓] Publish module contexts to separate XML files

Publishing

Timeouts

Ports

Modify the server ports.

Port Name	Port Numb
Tomcat admin port	8005
HTTP/1.1	8090
AJP/1.3	8009

MIME Mappings

2. Configuration in server.xml

- Add the same **`<Context>`** content in **`server.xml`** file found in **`conf`** folder of tomcat.
  - Move right down the file until you find **`</Host>`**
  - Add the following just above the **`</Host>`** tag

```
<Context docBase="/SRASYSWEB"  path="SRASYSWEB"
                        reloadable="true">
    <Resource name="jdbc/Test"     auth="Container"
    type="javax.sql.DataSource"
         maxActive="100"    maxIdle="30" maxWait="-1"
         username="root"    password="root"
         driverClass="oracle.jdbc.driver.OracleDriver"
         url="jdbc:oracle:thin:@127.0.0.1:1521:xe">
       </Resource>
</Context>
```

Add <resource-ref> in web.xml

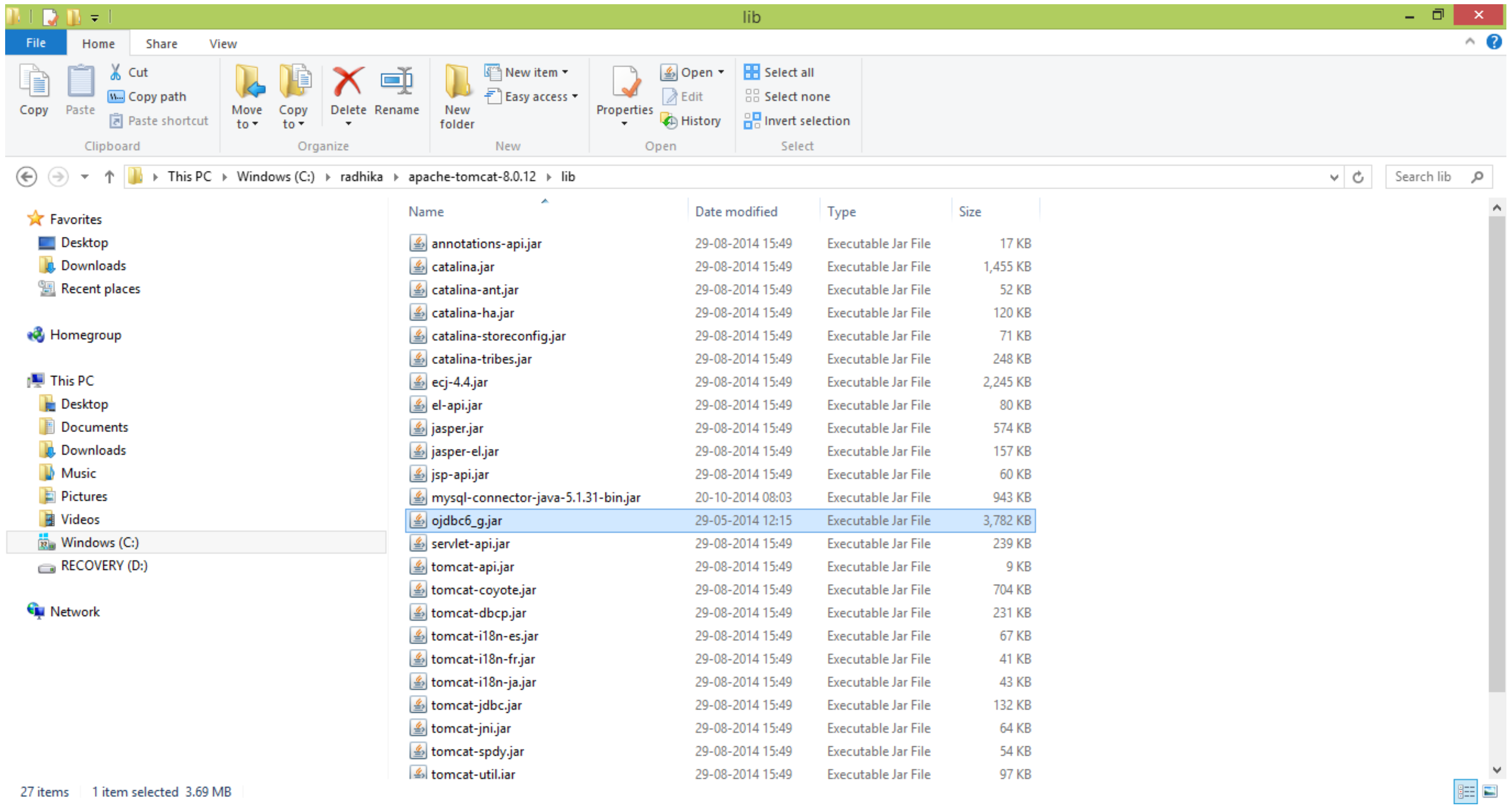
- In web.xml add the following

```
 <resource-ref>
    <description>Test</description>
    <res-ref-name>jdbc/Test</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

This tells which JNDI resources will our web application access.

Add libraries

- Copy the jar file for Oracle in the and drop it in the
  - `<Tomcat_base_folder>/lib` and
  - in the individual web application in WEB-INF/lib folder



Name	Date modified	Type	Size
annotations-api.jar	29-08-2014 15:49	Executable Jar File	17 KB
catalina.jar	29-08-2014 15:49	Executable Jar File	1,455 KB
catalina-ant.jar	29-08-2014 15:49	Executable Jar File	52 KB
catalina-ha.jar	29-08-2014 15:49	Executable Jar File	120 KB
catalina-storeconfig.jar	29-08-2014 15:49	Executable Jar File	71 KB
catalina-tribes.jar	29-08-2014 15:49	Executable Jar File	248 KB
ecj-4.4.jar	29-08-2014 15:49	Executable Jar File	2,245 KB
el-api.jar	29-08-2014 15:49	Executable Jar File	80 KB
jasper.jar	29-08-2014 15:49	Executable Jar File	574 KB
jasper-el.jar	29-08-2014 15:49	Executable Jar File	157 KB
jsp-api.jar	29-08-2014 15:49	Executable Jar File	60 KB
mysql-connector-java-5.1.31-bin.jar	20-10-2014 08:03	Executable Jar File	943 KB
ojdbc6_g.jar	29-05-2014 12:15	Executable Jar File	3,782 KB
servlet-api.jar	29-08-2014 15:49	Executable Jar File	239 KB
tomcat-api.jar	29-08-2014 15:49	Executable Jar File	9 KB
tomcat-coyote.jar	29-08-2014 15:49	Executable Jar File	704 KB
tomcat-dbcp.jar	29-08-2014 15:49	Executable Jar File	231 KB
tomcat-i18n-es.jar	29-08-2014 15:49	Executable Jar File	67 KB
tomcat-i18n-fr.jar	29-08-2014 15:49	Executable Jar File	41 KB
tomcat-i18n-ja.jar	29-08-2014 15:49	Executable Jar File	43 KB
tomcat-jdbc.jar	29-08-2014 15:49	Executable Jar File	132 KB
tomcat-jni.jar	29-08-2014 15:49	Executable Jar File	64 KB
tomcat-spdy.jar	29-08-2014 15:49	Executable Jar File	54 KB
tomcat-util.jar	29-08-2014 15:49	Executable Jar File	97 KB

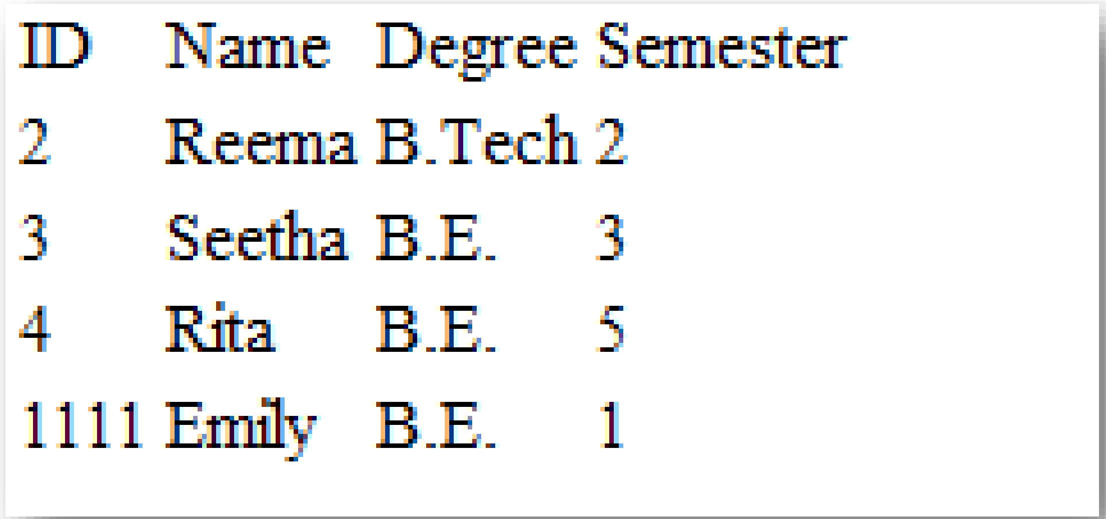
27 items   1 item selected 3.69 MB

Write code to connect to the database

```
package test;
import java.io.*;
import java.sql.*;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

@WebServlet("/JNDITest")
public class JNDITest extends HttpServlet {
private static final long serialVersionUID = 1L;
```

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
      Connection conn = null;
      PrintWriter out = response.getWriter();
      out.println("<html><head><title>JNDI test
      </title></head><body>");

      try {
      /* get the DataSource from using the JNDI name */
            InitialContext ctx = new InitialContext();
            DataSource ds = (DataSource)
            ctx.lookup("java:comp/env/jdbc/Test");

      /* Create connection and then continue as usual
            other JDBC calls */
      Statement s= conn.createStatement();
      ResultSet rs=s.executeQuery("SELECT * FROM
                                 STUDENT");
```

```
    out.println("<table><tr><td>ID</td><td>Name</td><
        td>Degree</td><td> Semester</td></tr> ");
        while (rs.next() ) {
        out.println("<tr><td>"+rs.getInt(1)
+"</td><td>"+rs.getString(2)+"</td><td>"+rs.getString(3
)+"</td><td>"+rs.getInt(4)+"</td></tr>");
}
        out.println("</table></body></html>");
        } catch (Exception e){
              out.println("Failed"+ e);
}}}
```



ID	Name	Degree	Semester
2	Reema	B.Tech	2
3	Seetha	B.E.	3
4	Rita	B.E.	5
1111	Emily	B.E.	1

Best Practices

- In case of **`DataSource`**, the DataSource JNDI name such as **`/jdbc/Test`** must be saved in **`web.xml`** <**`context-param> or <init-param>`** instead of hard coding this in the servlet.
- While for a small, infrequent database access, direct JDBC calls are fine, but for database intensive application, DataSource must be used.
- Today's application connect to the database using technologies like EJB and Hibernate, where all the JDBC calls are handled by the application server and the 3rd party classes itself. A developer needs to create just a simple Java Bean classes that will act as entities of the system.

Summary

- Connecting to the database from servlet can be done two ways - Connect to database directly or using DataSources.
- The Oracle ojdbc jar file must be added in the WEB-INF\lib. This can be done by simply copying the file and dropping into the project into eclipse IDE.
- DataSource interface is used as an alternative to DriverManager of JDBC 2.0 core package.
- DataSource object is created when the details like the URL of the data source and the Driver name is provided.
- With Connection object obtained from DataSource, distributed transaction is automatically supported.