Connection Pooling

The Java 2 Enterprise Edition (J2EE) specification provides a distributed services-based architecture for implementing highly scalable, reliable applications. In general, a J2EE application architecture maps to the Model-View-Controller (MVC) framework.

A typical business application use case would be realized by components in all the three layers on the server side. Given the large number of user interactions (millions for customer-facing applications), the finite server-side resources need to be optimally shared. Such resources may include databases, message queues, directories, enterprise systems (SAP, CICS), and so forth, each of which is accessed by an application using a connection object that represents the resource entry point. Managing access to those shared resources is essential for meeting the high-performance requirements for J2EE applications.

The process of creating a connection, always an expensive, time-consuming operation, is multiplied in these environments where a large number of users are accessing the database. Creating connections over and over in these environments is too expensive and reduce performance of the application.

Connection pooling is a technique that was pioneered by database vendors to allow multiple clients to share a cached set of connection objects that provide access to a database resource. Connection pooling has become the standard for middleware database drivers.

What is Connection Pooling?

A cache of database connections maintained in the pool memory, so that the connections can be reused when user request for the connection

Particularly for server-side web applications, a connection pool is the standard way to maintain a “pool” of active database connections in memory which are reused across requests.

JNDI

The Java Naming and Directory Interface (JNDI) is an application programming interface (API) for accessing different kinds of naming and directory services. JNDI is not specific to a particular naming or directory service, it can be used to access many different kinds of systems including file systems; distributed objects systems like CORBA, Java RMI, and EJB; and directory services like LDAP, Novell NetWare, and NIS+.

JNDI is similar to JDBC in that they are both Object-Oriented Java APIs that provide a common abstraction for accessing services from different vendors. While JDBC can be used to access a variety of relational databases, JNDI can be used to access a variety of of naming and directory services.

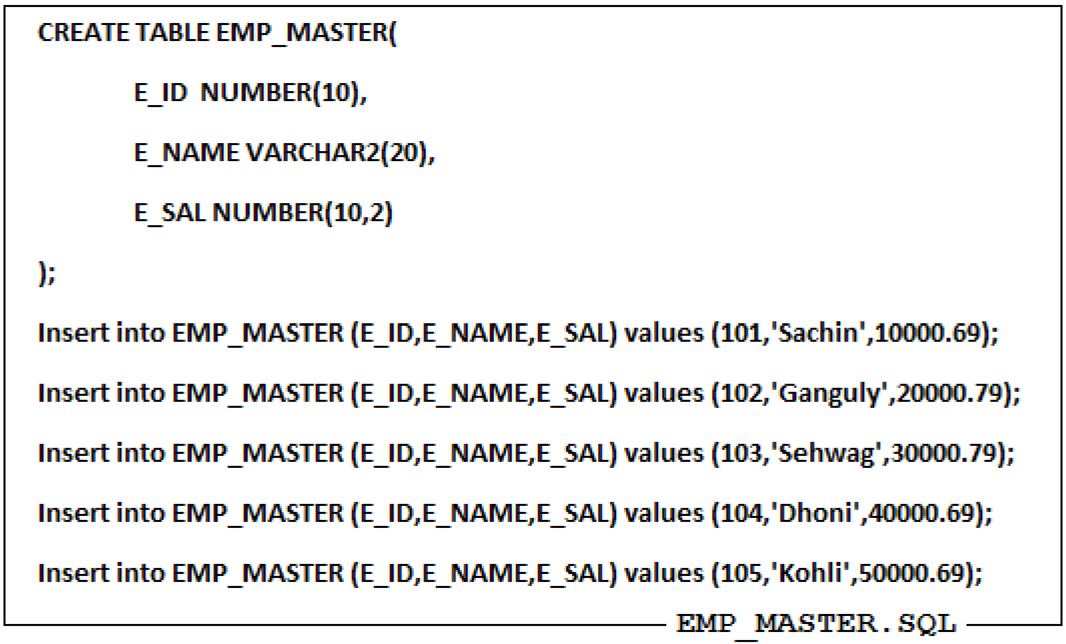
The benefits of using a JNDI DataSource

There are major advantages to connecting to a data source using a DataSource object registered with a JNDI naming service rather than using the DriverManager facility.

- The first is that it makes code more portable. (With the DriverManager, the name of a JDBC driver class, which usually identifies a particular driver vendor, is included in application code. This makes the application specific to that vendor's driver product and thus non-portable)
- It makes code much easier to maintain. If any of the necessary information about the data source changes, only the relevant DataSource properties need to be modified
- Multiple wars can use the same connection pool which might be better use of resources
- It's a fine way to externalize configuration.

Configuring JNDI DataSource for Database Connection Pooling in Tomcat

Step 1: Create a table in Oracle Database and insert the few records using below SQL Queries

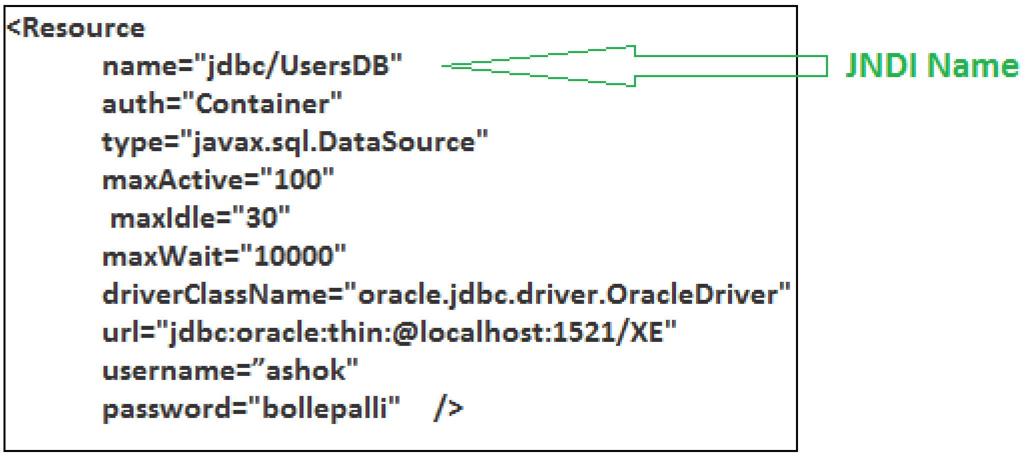


```sql
CREATE TABLE EMP_MASTER(

        E_ID  NUMBER(10),

        E_NAME VARCHAR2(20),

        E_SAL NUMBER(10,2)

);

Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (101,'Sachin',10000.69);

Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (102,'Ganguly',20000.79);

Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (103,'Sehwag',30000.79);

Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (104,'Dhoni',40000.69);

Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (105,'Kohli',50000.69);
```

EMP_MASTER.SQL

Step 2: Configuring Context Details

To declare a JNDI DataSource for the Oracle database, create a **Resource** XML element with the following content



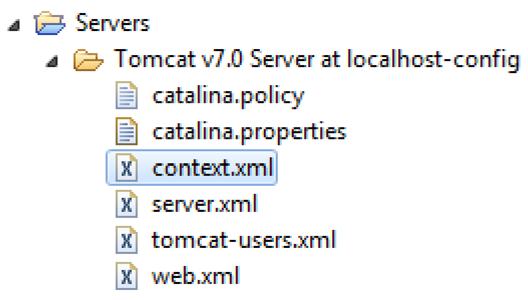
```xml
<Resource
        name="jdbc/UsersDB"
        auth="Container"
        type="javax.sql.DataSource"
        maxActive="100"
         maxIdle="30"
        maxWait="10000"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@localhost:1521/XE"
        username="ashok"
        password="bollepalli"    />
```

Note: Change Database Username and password according to your DB details

Add this element inside the root element <Context> in a **context.xml** file. There are two places where the context.xml file can reside (create one if not exist):

1. Inside **/META-INF** directory of a web application: the JNDI DataSource is only available to the application itself, thus it cannot be shared among other ones. In addition, this makes the configuration dependent on the application.
2. Inside **$CATALINA_BASE/conf** directory (server conf folder): this is the preferred place because the JNDI DataSource will be available to all web applications and it's independent of any applications.

If you are using Tomcat inside Eclipse IDE, you need to modify the context.xml file under the Servers project. That is because Eclipse made a copy of Tomcat configuration:



The following table describes the attributes specified in the above configuration:

Attribute name	Description
name	Name of the resource.
auth	Specify authentication mechanism for the application code, can be Application or Container.
type	The fully qualified Java class name expected by the web application when it performs a lookup for this resource.
maxActive	Maximum number of database connections in pool. Set to -1 for no limit.
maxIdle	Maximum number of idle database connections to retain in pool. Set to -1 for no limit.
maxWait	Maximum time to wait for a database connection to become available in ms, in this example 10 seconds. An Exception is thrown if this timeout is exceeded. Set to -1 to wait indefinitely.
driverClassName	The fully qualified Java class name of the database driver
url	The JDBC connection URL.
username	Database user name.
password	Database user password.

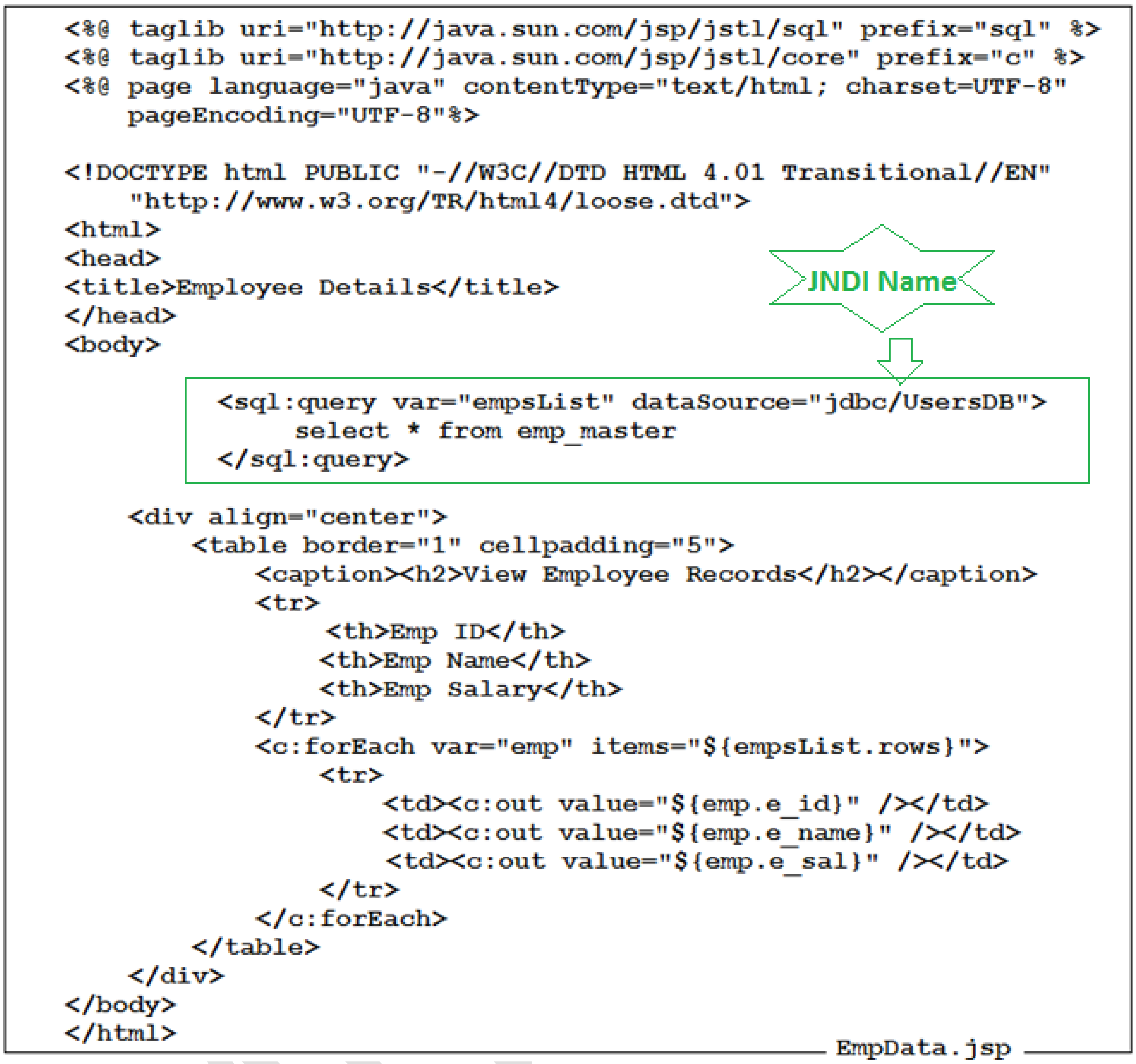
Step 3: Configuring resource ref details in project web.xml file

Add the following declaration in project web.xml file

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

This is necessary in order to make the JNDI DataSource available to the application under the specified namespace jdbc/UsersDB

Step 4: Create JSP Page in web application to retrieve Employees data from Database with JNDI DataSource



```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Employee Details</title>
</head>
<body>

        <sql:query var="empsList" dataSource="jdbc/UsersDB">
             select * from emp_master
        </sql:query>

    <div align="center">
        <table border="1" cellpadding="5">
            <caption><h2>View Employee Records</h2></caption>
            <tr>
                <th>Emp ID</th>
                <th>Emp Name</th>
                <th>Emp Salary</th>
            </tr>
            <c:forEach var="emp" items="${empsList.rows}">
                <tr>
                    <td><c:out value="${emp.e_id}" /></td>
                    <td><c:out value="${emp.e_name}" /></td>
                    <td><c:out value="${emp.e_sal}" /></td>
                </tr>
            </c:forEach>
        </table>
    </div>
</body>
</html>
```

EmpData.jsp

Note: Here, we use the JSTL's SQL tag **query** to make a SELECT query to the database. Note that the **DataSource** attribute refers to the JNDI resource name declared in the web.xml file.

Using JNDI DataSource in Java Program

We can look up the configured JNDI DataSource using Java code as follows:

```
Context initContext = new InitialContext();
Context envContext = (Context) initContext.lookup("java:comp/env");
DataSource ds = (DataSource) envContext.lookup("jdbc/UsersDB");
Connection conn = ds.getConnection();
```

After obtaining the connection, we can write JDBC code using this conn object

Getting JNDI DataSource using annotations

Alternatively, we can use the **@Resource** annotation (**javax.annotation.Resource**) instead of the lookup code above. For example, declare a field called **dataSource** in the servlet like this:

```
@Resource(name = "jdbc/UsersDB")
private DataSource dataSource;
```

Tomcat will look up the specified resource name and inject an actual implementation when it discovers this annotation. Therefore, the servlet code looks like this:

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

    @Resource(name = "jdbc/UsersDB")
    private DataSource dataSource;              //Getting JNDI DataSource

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
             PrintWriter writer = response.getWriter();
        try {
            Connection conn = dataSource.getConnection(); //getting Connection

            Statement statement = conn.createStatement();
            String sql = "select * from emp_master";
            ResultSet rs = statement.executeQuery(sql);

            while (rs.next()) {
                writer.write(rs.getInt("e_id");
                writer.write(rs.getInt("e_name");
                writer.write(rs.getInt("e_sal");
                writer.write(rs.getInt("<br/>");
            }
        } catch (SQLException ex) {
            System.err.println(ex);
        }
    }
}
```

EmpListServlet.java

=====Happy Learning====