# Hibernate Tomcat JNDI DataSource Example Tutorial

We have already seen how to use Hibernate ORM tool in standalone java application, today we will learn how to use **Hibernate with DataSource** in Tomcat servlet container.

Using hibernate in web application is very easy, all we need is to configure `DataSource` properties in hibernate configuration file. First of all we need to setup test database and JNDI DataSource in tomcat container.

## Database Setup

I am using MySQL for my example, below script is executed to create a simple table and insert some values into it.

employee.sql

```
CREATE TABLE `Employee` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  `role` varchar(20) DEFAULT NULL,
  `insert_time` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=26 DEFAULT CHARSET=utf8;

INSERT INTO `Employee` (`id`, `name`, `role`, `insert_time`)
VALUES
    (3, 'Pankaj', 'CEO', now());
INSERT INTO `Employee` (`id`, `name`, `role`, `insert_time`)
VALUES
    (14, 'David', 'Developer', now());
```

The Database schema name is **TestDB**.

## Tomcat JNDI DataSource Configuration

For configuring tomcat container to initialize DataSource, we need to make some changes in tomcat server.xml and context.xml files.

server.xml

```
<Resource name="jdbc/MyLocalDB"
      global="jdbc/MyLocalDB"
      auth="Container"
      type="javax.sql.DataSource"
      driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://localhost:3306/TestDB"
      username="pankaj"
      password="pankaj123"

      maxActive="100"
```

```
          maxIdle="20"
          minIdle="5"
          maxWait="10000"/>
```

Add above resource in the server.xml `GlobalNamingResources` element.

context.xml

```
<ResourceLink name="jdbc/MyLocalDB"
                global="jdbc/MyLocalDB"
                auth="Container"
                type="javax.sql.DataSource" />
```

Add above `ResourceLink` in the context.xml file, it's required so that applications can access the JNDI resource with name `jdbc/MyLocalDB`.

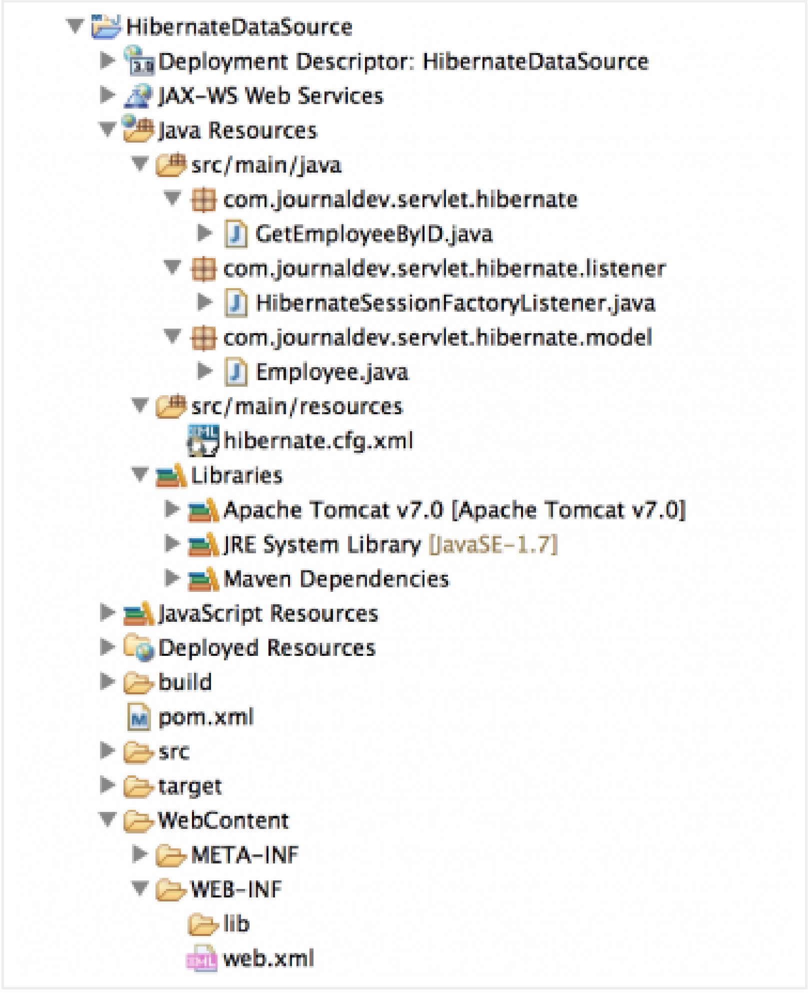
Just restart the server, you should not see any errors in the tomcat server logs. If there are any wrong configurations, such as password is wrong, you will get the corresponding exception in the server log.

You also need to make sure that MySQL driver jar file is inside the tomcat lib directory, otherwise tomcat will not be able to create database connection and you will get `ClassNotFoundException` in logs.

Now our database and tomcat server JNDI setup is ready, let's move to create our web application using hibernate.

## Hibernate Dynamic Web Project

Create a dynamic web project in Eclipse and then configure it as Maven project. Our final project structure will look like below image.



```
HibernateDataSource
  Deployment Descriptor: HibernateDataSource
  JAX-WS Web Services
  Java Resources
    src/main/java
      com.journaldev.servlet.hibernate
        GetEmployeeByID.java
      com.journaldev.servlet.hibernate.listener
        HibernateSessionFactoryListener.java
      com.journaldev.servlet.hibernate.model
        Employee.java
    src/main/resources
      hibernate.cfg.xml
    Libraries
      Apache Tomcat v7.0 [Apache Tomcat v7.0]
      JRE System Library [JavaSE-1.7]
      Maven Dependencies
  JavaScript Resources
  Deployed Resources
  build
  pom.xml
  src
  target
  WebContent
    META-INF
    WEB-INF
      lib
      web.xml
```

Note that I am using **Tomcat-7** for my project deployment and I have added it to the build path, so that we don't need to separately add Servlet API dependencies in our project.

Tomcat-7 supports **Servlet 3 specs** and we will be using annotations to create our servlets. If you are not familiar with Servlet 3 annotations, you should check out Servlet Tutorial for Beginners.

Let's look into each of the components one by one.

## Maven Dependencies

Our final pom.xml file looks like below.

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.o
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache
    <modelVersion>4.0.0</modelVersion>
    <groupId>HibernateDataSource</groupId>
    <artifactId>HibernateDataSource</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <dependencies>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>4.3.5.Final</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.0.5</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.3</version>
                <configuration>
                    <warSourceDirectory>WebContent</warSourceDirectory>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                </configuration>
            </plugin>
        </plugins>
        <finalName>${project.artifactId}</finalName>
    </build>
</project>
```

I am using Hibernate latest version **4.3.5.Final**, `hibernate-core` dependency is added for Hibernate. `mysql-connector-java` dependency is added because we are using MySQL database, although scope is provided because it's already part of the tomcat container libraries.

Even if we don't add MySQL driver dependencies, our project will compile and run fine. However it's better to include it so that if someone will look into the project dependencies, it will be clear that we are using MySQL database.

## Hibernate DataSource Configuration

Our hibernate configuration file with datasource looks like below.

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Dri
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect
        <property name="hibernate.connection.datasource">java:comp/env/jdbc/M
        <property name="hibernate.current_session_context_class">thread</prop

        <!-- Mapping with model class containing annotations -->
    <mapping class="com.journaldev.servlet.hibernate.model.Employee"/>
    </session-factory>
</hibernate-configuration>
```

`hibernate.connection.datasource` property is used to provide the DataSource name that will be used by Hibernate for database operations.

## Model Class

As you can see in hibernate configuration file, we are using annotations in our model class Employee. Our model bean looks like below.

Employee.java

```java
package com.journaldev.servlet.hibernate.model;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

@Entity
@Table(name="Employee", 
       uniqueConstraints={@UniqueConstraint(columnNames={"ID"})})
public class Employee {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="ID", nullable=false, unique=true, length=11)
    private int id;
    
    @Column(name="NAME", length=20, nullable=true)
    private String name;
    
    @Column(name="ROLE", length=20, nullable=true)
    private String role;
    
    @Column(name="insert_time", nullable=true)
    private Date insertTime;
    
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getRole() {
        return role;
    }
    public void setRole(String role) {
        this.role = role;
    }
    public Date getInsertTime() {
        return insertTime;
    }
    public void setInsertTime(Date insertTime) {
        this.insertTime = insertTime;
    }
}
```

Model bean is same as we used in Hibernate Beginners Tutorial, you should check it out if you have any confusion related to any of the annotations used.

## Servlet Listener

Since we have to initialize Hibernate `SessionFactory` because we can use it in the application and also when web application is destroyed, we need to destroy SessionFactory. So the best place to do this in a `ServletContextListener` implementation.

HibernateSessionFactoryListener.java

```java
package com.journaldev.servlet.hibernate.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;
import org.jboss.logging.Logger;

@WebListener
public class HibernateSessionFactoryListener implements ServletContextListene

    public final Logger logger = Logger.getLogger(HibernateSessionFactoryList

    public void contextDestroyed(ServletContextEvent servletContextEvent) {
        SessionFactory sessionFactory = (SessionFactory) servletContextEvent.
        if(sessionFactory != null && !sessionFactory.isClosed()){
            logger.info("Closing sessionFactory");
            sessionFactory.close();
        }
        logger.info("Released Hibernate sessionFactory resource");
    }

    public void contextInitialized(ServletContextEvent servletContextEvent) {
        Configuration configuration = new Configuration();
        configuration.configure("hibernate.cfg.xml");
        logger.info("Hibernate Configuration created successfully");

        ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder(
        logger.info("ServiceRegistry created successfully");
        SessionFactory sessionFactory = configuration
                .buildSessionFactory(serviceRegistry);
        logger.info("SessionFactory created successfully");

        servletContextEvent.getServletContext().setAttribute("SessionFactory"
        logger.info("Hibernate SessionFactory Configured successfully");
    }

}
```

If you are not familiar with servlet listeners, please read Servlet Listener Tutorial.

## Servlet Implementation

Let’s write a simple servlet where we will pass employee id as request parameter and it will print out the employee information from database, obviously we will use Hibernate to query the database and get employee information.

GetEmployeeByID.java

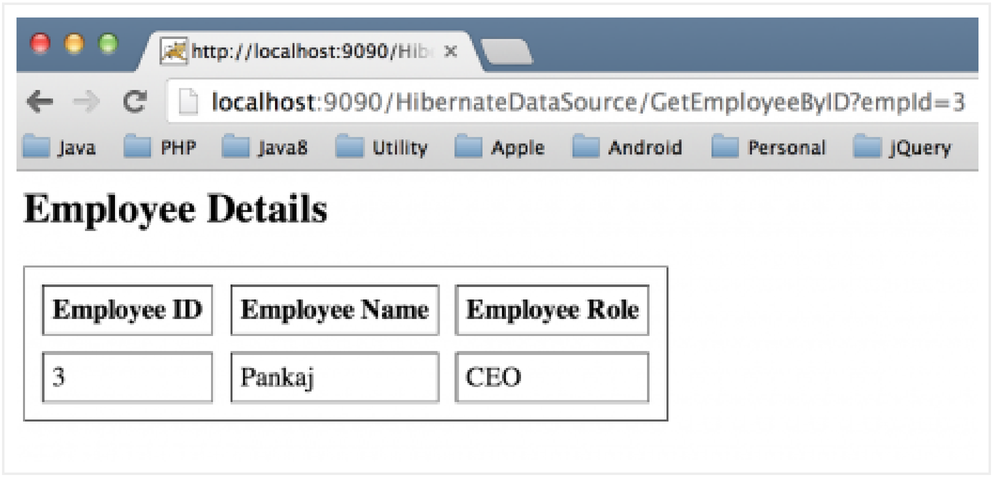
```java
package com.journaldev.servlet.hibernate;

import java.io.IOException;
import java.io.PrintWriter;

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 org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.jboss.logging.Logger;

import com.journaldev.servlet.hibernate.model.Employee;

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

    public final Logger logger = Logger.getLogger(GetEmployeeByID.class);

    protected void doGet(HttpServletRequest request, HttpServletResponse respo
        int empId = Integer.parseInt(request.getParameter("empId"));
        logger.info("Request Param empId="+empId);

        SessionFactory sessionFactory = (SessionFactory) request.getServletCo

        Session session = sessionFactory.getCurrentSession();
        Transaction tx = session.beginTransaction();
        Employee emp = (Employee) session.get(Employee.class, empId);
        tx.commit();
        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        if(emp != null){
        out.print("<html><body><h2>Employee Details</h2>");
        out.print("<table border=\"1\" cellspacing=10 cellpadding=5>");
        out.print("<th>Employee ID</th>");
        out.print("<th>Employee Name</th>");
        out.print("<th>Employee Role</th>");

            out.print("<tr>");
            out.print("<td>" + empId + "</td>");
```

```
                out.print("<td>" + emp.getName() + "</td>");
                out.print("<td>" + emp.getRole() + "</td>");
                out.print("</tr>");
            out.print("</table></body><br/>");

            out.print("</html>");
            }else{
                out.print("<html><body><h2>No Employee Found with ID="+empId+"</h
            }
        }

    }
```

It’s a very simple servlet class, I am using `@WebServlet` annotation to provide the URI pattern for it.

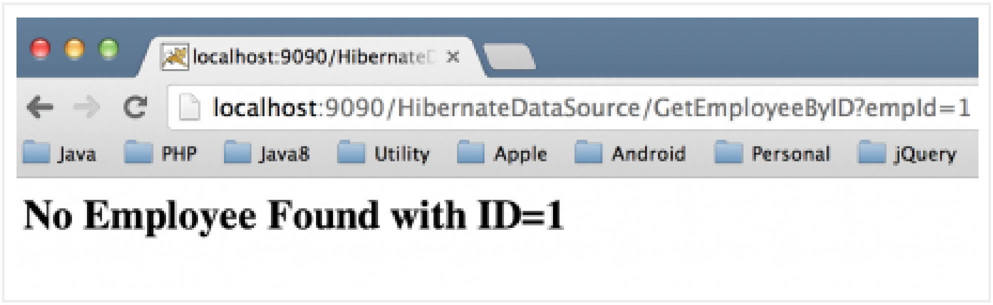
## Testing Web Application

Our application is ready now, just export as war file and deploy it in the tomcat container. Below are some of the screenshots when we invoke our application servlet.


http://localhost:9090/Hib
localhost:9090/HibernateDataSource/GetEmployeeByID?empId=3
Java
PHP
Java8
Utility
Apple
Android
Personal
jQuery

Employee Details

| Employee ID | Employee Name | Employee Role |
|---|---|---|
| 3 | Pankaj | CEO |



No Employee Found with ID=1

Notice that I am passing **empId** request parameter in the request URL query string. You will also see our application generated logs in the server logs.

```
May 08, 2014 8:14:16 PM org.hibernate.cfg.Configuration configure
INFO: HHH000043: Configuring from resource: hibernate.cfg.xml
May 08, 2014 8:14:16 PM org.hibernate.cfg.Configuration getConfigurationInput
INFO: HHH000040: Configuration resource: hibernate.cfg.xml
May 08, 2014 8:14:16 PM org.hibernate.cfg.Configuration doConfigure
INFO: HHH000041: Configured SessionFactory: null
May 08, 2014 8:14:16 PM com.journaldev.servlet.hibernate.listener.HibernateSe
INFO: Hibernate Configuration created successfully
May 08, 2014 8:14:16 PM com.journaldev.servlet.hibernate.listener.HibernateSe
INFO: ServiceRegistry created successfully
```

```
May 08, 2014 8:14:16 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
May 08, 2014 8:14:17 PM org.hibernate.engine.jdbc.internal.LobCreatorBuilder u
INFO: HHH000423: Disabling contextual LOB creation as JDBC driver reported JDB
May 08, 2014 8:14:17 PM org.hibernate.engine.transaction.internal.TransactionF
INFO: HHH000399: Using default transaction strategy (direct JDBC transactions)
May 08, 2014 8:14:17 PM org.hibernate.hql.internal.ast.ASTQueryTranslatorFacto
INFO: HHH000397: Using ASTQueryTranslatorFactory
May 08, 2014 8:14:17 PM com.journaldev.servlet.hibernate.listener.HibernateSes
INFO: SessionFactory created successfully
May 08, 2014 8:14:17 PM com.journaldev.servlet.hibernate.listener.HibernateSes
INFO: Hibernate SessionFactory Configured successfully
May 08, 2014 8:14:32 PM com.journaldev.servlet.hibernate.GetEmployeeByID doGet
INFO: Request Param empId=3
May 08, 2014 8:15:22 PM com.journaldev.servlet.hibernate.GetEmployeeByID doGet
INFO: Request Param empId=3
```

If you will undeploy the application or stop the server, you will see server logs for destroying the SessionFactory.

```
May 08, 2014 11:31:16 PM com.journaldev.servlet.hibernate.listener.HibernateSes
INFO: Closing sessionFactory
May 08, 2014 11:31:16 PM com.journaldev.servlet.hibernate.listener.HibernateSes
INFO: Released Hibernate sessionFactory resource
```

That's all for **Hibernate DataSource example** for tomcat container, I hope it's easy to understand and implement. Download the sample project from below link and play around with it to learn more.

**Hibernate DataSource Project**
701 downloads