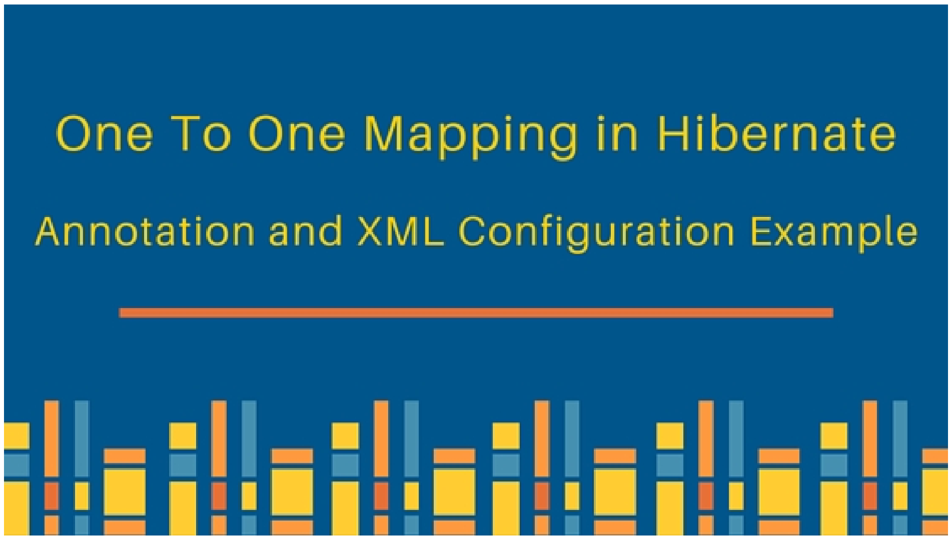
Hibernate One to One Mapping Example Annotation

APRIL 2, 2018 BY PANKAJ — 15 COMMENTS

Today we will look into One to One Mapping in Hibernate. We will look into Hibernate One To One Mapping example using Annotation and XML configuration.

Table of Contents [hide]

One to One Mapping in Hibernate


One To One Mapping in Hibernate
Annotation and XML Configuration Example

Most of the times, database tables are associated with each other. There are many forms of association – **one-to-one**, **one-to-many** and **many-to-many** are at the broad level. These can be further divided into unidirectional and bidirectional mappings. Today we will look into implementing **Hibernate One to One Mapping** using **XML configuration** as well as using **annotation configuration**.

Hibernate One to One Mapping Example Database Setup

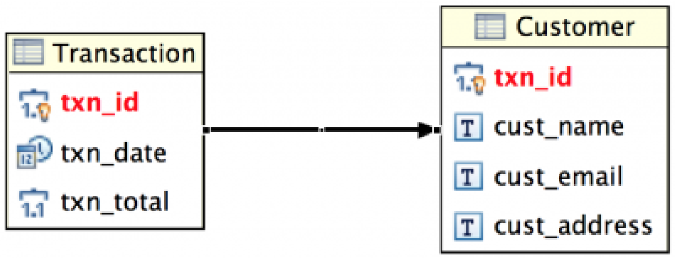
First of all we would need to setup One to One mapping in database tables. We will create two tables for our example – Transaction and Customer. Both of these tables will have one to one mapping. Transaction will be the primary table and we will be using **Foreign Key** in Customer table for one-to-one mapping.

I am providing MySQL script, that is the database I am using for this tutorial. If you are using any other database, make sure to change the script accordingly.

```
-- Create Transaction Table
CREATE TABLE `Transaction` (
  `txn_id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `txn_date` date NOT NULL,
  `txn_total` decimal(10,0) NOT NULL,
  PRIMARY KEY (`txn_id`)
) ENGINE=InnoDB AUTO_INCREMENT=16 DEFAULT CHARSET=utf8;

-- Create Customer table
CREATE TABLE `Customer` (
  `txn_id` int(11) unsigned NOT NULL,
  `cust_name` varchar(20) NOT NULL DEFAULT '',
  `cust_email` varchar(20) DEFAULT NULL,
  `cust_address` varchar(50) NOT NULL DEFAULT '',
  PRIMARY KEY (`txn_id`),
  CONSTRAINT `customer_ibfk_1` FOREIGN KEY (`txn_id`) REFERENCES `Transaction`
(`txn_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

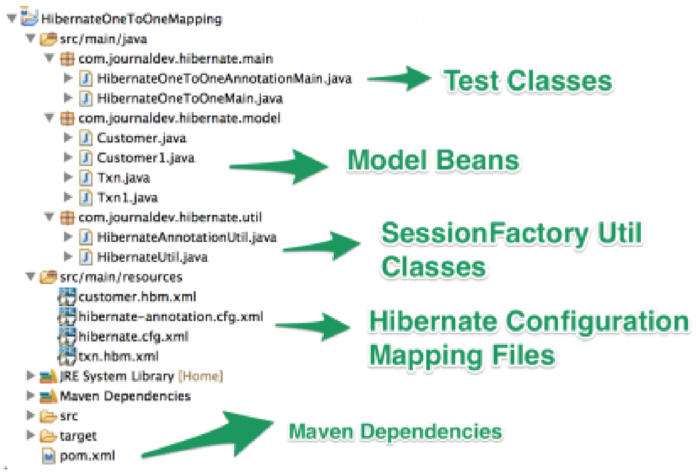
Entity Relation Diagram (ERD) of above one-to-one mapping between tables looks like below image.



Our database setup is ready, let's move on the Hibernate One to One Example Project now.

Hibernate One to One Mapping Example Project Structure

Create a simple Maven project in your Java IDE, I am using Eclipse. Our final project structure will look like below image.


HibernateOneToOneMapping
src/main/java
com.journaldev.hibernate.main
HibernateOneToOneAnnotationMain.java
HibernateOneToOneMain.java
com.journaldev.hibernate.model
Customer.java
Customer1.java
Txn.java
Txn1.java
com.journaldev.hibernate.util
HibernateAnnotationUtil.java
HibernateUtil.java
src/main/resources
customer.hbm.xml
hibernate-annotation.cfg.xml
hibernate.cfg.xml
txn.hbm.xml
JRE System Library [Home]
Maven Dependencies
src
target
pom.xml
Test Classes
Model Beans
SessionFactory Util Classes
Hibernate Configuration Mapping Files
Maven Dependencies

First of all we will look into XML Based Hibernate One to One Mapping example and then we will implement the same thing using annotation.

Hibernate Maven Dependencies

Our final pom.xml file looks like below.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.journaldev.hibernate</groupId>
  <artifactId>HibernateOneToOneMapping</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <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>
        </dependency>
  </dependencies>
</project>
```

Dependencies are just for hibernate and mysql java driver. Note that I am using Hibernate latest version **4.3.5.Final** and MySQL java driver based on my MySQL database server version (5.0.5).

Hibernate 4 uses JBoss logging and it gets imported automatically as transitive dependency. You can confirm it in the maven dependencies of the project. If you are using Hibernate older versions, you might have to add slf4j dependencies.

Hibernate One to One Mapping Model Classes

Model classes for Hibernate One to One mapping to reflect database tables would be like below.

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

```
import java.util.Date;

public class Txn {

        private long id;
        private Date date;
        private double total;
        private Customer customer;

        @Override
        public String toString(){
                return id+", "+total+", "+customer.getName()+", 
"+customer.getEmail()+", "+customer.getAddress();
        }
        public long getId() {
                return id;
        }
        public void setId(long id) {
                this.id = id;
        }
```

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

public class Customer {

        private long id;
        private String name;
        private String email;
        private String address;

        private Txn txn;

        public long getId() {
                return id;
        }
        public void setId(long id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
```

Since we are using XML based configuration for mapping, above model classes are simple POJO classes or Java Beans with getter-setter methods. I am using class name as `Txn` to avoid confusion because Hibernate API have a class name as `Transaction`.

Hibernate One to One Mapping Configuration

Let's create hibernate one to one mapping configuration files for Txn and Customer tables.

`txn.hbm.xml`

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
        <class name="com.journaldev.hibernate.model.Txn" table="TRANSACTION" >
                <id name="id" type="long">
                        <column name="txn_id" />
                        <generator class="identity" />
                </id>
                <property name="date" type="date">
                        <column name="txn_date" />
                </property>
                <property name="total" type="double">
                        <column name="txn_total" />
                </property>
                <one-to-one name="customer"
class="com.journaldev.hibernate.model.Customer"
                        cascade="save-update" />
        </class>

</hibernate-mapping>
```

The important point to note above is the hibernate `one-to-one` element for customer property.

`customer.hbm.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-mapping-3.0.dtd" >
<hibernate-mapping>

        <class name="com.journaldev.hibernate.model.Customer" table="CUSTOMER">
```

```
                <id name="id" type="long">
                        <column name="txn_id" />
                        <generator class="foreign">
                                <param name="property">txn</param>
                        </generator>
                </id>
                <one-to-one name="txn" class="com.journaldev.hibernate.model.Txn"
                        constrained="true"></one-to-one>

                <property name="name" type="string">
                        <column name="cust_name"></column>
                </property>
                <property name="email" type="string">
                        <column name="cust_email"></column>
                </property>
```

generator class="foreign" is the important part that is used for hibernate **foreign key** implementation.

Hibernate Configuration File

Here is the hibernate configuration file for XML based hibernate mapping configuration.

`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.Driver</property>
        <property name="hibernate.connection.password">pankaj123</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost/TestDB</property>
        <property name="hibernate.connection.username">pankaj</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <property name="hibernate.current_session_context_class">thread</property>
        <property name="hibernate.show_sql">true</property>

        <mapping resource="txn.hbm.xml"/>
        <mapping resource="customer.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Hibernate configuration file is simple, it has database connection properties and hibernate mapping resources.

Hibernate SessionFactory Utility

Here is the utility class to create hibernate SessionFactory instance.

```
package com.journaldev.hibernate.util;

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

public class HibernateUtil {

        private static SessionFactory sessionFactory;

        private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
                Configuration configuration = new Configuration();
                configuration.configure("hibernate.cfg.xml");
                System.out.println("Hibernate Configuration loaded");

                ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();
```

That's it, lets write a test program to test the hibernate one to one mapping xml based configuration.

Hibernate One to One Mapping XML Configuration Test Program

In the hibernate one to one mapping example test program, first we will create Txn object and save it. Once it's saved into database, we will use the generated id to retrieve the Txn object and print it.

```
package com.journaldev.hibernate.main;

import java.util.Date;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.journaldev.hibernate.model.Customer;
import com.journaldev.hibernate.model.Txn;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateOneToOneMain {

        public static void main(String[] args) {

                Txn txn = buildDemoTransaction();

                SessionFactory sessionFactory = null;
                Session session = null;
                Transaction tx = null;
                try{
```

Now when we run above one to one mapping in hibernate test program, we get following output.

```
Hibernate Configuration loaded
Hibernate serviceRegistry created
Session created
Hibernate: insert into TRANSACTION (txn_date, txn_total) values (?, ?)
Hibernate: insert into CUSTOMER (cust_name, cust_email, cust_address, txn_id) values
(?, ?, ?, ?)
Transaction ID=19
Hibernate: select txn0_.txn_id as txn_id1_1_0_, txn0_.txn_date as txn_date2_1_0_,
txn0_.txn_total as txn_tota3_1_0_,
customer1_.txn_id as txn_id1_0_1_, customer1_.cust_name as cust_nam2_0_1_,
customer1_.cust_email as cust_ema3_0_1_,
customer1_.cust_address as cust_add4_0_1_ from TRANSACTION txn0_ left outer join
CUSTOMER customer1_ on
txn0_.txn_id=customer1_.txn_id where txn0_.txn_id=?
Transaction Details=
19, 100.0, Pankaj Kumar, pankaj@gmail.com, Bangalore, India
Closing SessionFactory
```

As you can see that it's working fine and we are able to retrieve data from both the tables using transaction id. Check the SQL used by Hibernate internally to get the data, its using joins to get the data from both the tables.

Hibernate One to One Mapping Annotation

In the above section, we saw how to use XML based configuration for hibernate one to one mapping, now let's see how we can use JPA and Hibernate annotation to achieve the same thing.

Hibernate Configuration File

`hibernate-annotation.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.Driver</property>
        <property name="hibernate.connection.password">pankaj123</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost/TestDB</property>
        <property name="hibernate.connection.username">pankaj</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

        <property name="hibernate.current_session_context_class">thread</property>
        <property name="hibernate.show_sql">true</property>

        <mapping class="com.journaldev.hibernate.model.Txn1"/>
        <mapping class="com.journaldev.hibernate.model.Customer1"/>
    </session-factory>
</hibernate-configuration>
```

Hibernate configuration is simple, as you can see that I have two model classes that we will use with annotations – `Txn1` and `Customer1`.

Hibernate One to One Mapping Annotation Example Model Classes

For hibernate one to one mapping annotation configuration, model classes are the most important part. Let's see how our model classes look.

```
package com.journaldev.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.OneToOne;
import javax.persistence.Table;

import org.hibernate.annotations.Cascade;

@Entity
@Table(name="TRANSACTION")
public class Txn1 {

        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        @Column(name="txn_id")
```

Notice that most of the annotations are from Java Persistence API because Hibernate provide it's implementation. However for cascading, we would need to use Hibernate annotation `org.hibernate.annotations.Cascade` and enum `org.hibernate.annotations.CascadeType`.

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

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
@Table(name="CUSTOMER")
public class Customer1 {

        @Id
        @Column(name="txn_id", unique=true, nullable=false)
        @GeneratedValue(generator="gen")
        @GenericGenerator(name="gen", strategy="foreign", parameters=
```

Note that we would need to `@GenericGenerator` so that id is used from the txn rather than generating it.

Hibernate SessionFactory Utility class

Creating SessionFactory is independent of the way we provide hibernate mapping. Our utility class for creating SessionFactory looks like below.

```
package com.journaldev.hibernate.util;

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

public class HibernateAnnotationUtil {

        private static SessionFactory sessionFactory;

        private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate-annotation.cfg.xml
                Configuration configuration = new Configuration();
                configuration.configure("hibernate-annotation.cfg.xml");
                System.out.println("Hibernate Annotation Configuration loaded");

                ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();
```

Hibernate One to One Mapping Annotation Example Test Program

Here is a simple test program for our hibernate one to one mapping annotation example.

```
package com.journaldev.hibernate.main;

import java.util.Date;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import com.journaldev.hibernate.model.Customer1;
import com.journaldev.hibernate.model.Txn1;
import com.journaldev.hibernate.util.HibernateAnnotationUtil;

public class HibernateOneToOneAnnotationMain {

        public static void main(String[] args) {

                Txn1 txn = buildDemoTransaction();

                SessionFactory sessionFactory = null;
                Session session = null;
                Transaction tx = null;
```

Here is the output snippet when we execute above program.

```
Hibernate Annotation Configuration loaded
Hibernate Annotation serviceRegistry created
Session created using annotations configuration
Hibernate: insert into TRANSACTION (txn_date, txn_total) values (?, ?)
Hibernate: insert into CUSTOMER (cust_address, cust_email, cust_name, txn_id) values
(?, ?, ?, ?)
Annotation Example. Transaction ID=20
Hibernate: select txn1x0_.txn_id as txn_id1_1_0_, txn1x0_.txn_date as txn_date2_1_0_,
txn1x0_.txn_total as txn_tota3_1_0_,
customer1x1_.txn_id as txn_id1_0_1_, customer1x1_.cust_address as cust_add2_0_1_,
customer1x1_.cust_email as cust_ema3_0_1_,
customer1x1_.cust_name as cust_nam4_0_1_ from TRANSACTION txn1x0_ left outer join
CUSTOMER customer1x1_ on
txn1x0_.txn_id=customer1x1_.txn_id where txn1x0_.txn_id=?
Annotation Example. Transaction Details=
20, 100.0, Pankaj Kr, pankaj@yahoo.com, San Jose, USA
Closing SessionFactory
```

Notice that the output is similar to hibernate one to one XML based configuration.

That's all for Hibernate One to One mapping example, you can download the final project from below link and play around with it to learn more.

Download Hibernate OneToOne Mapping Project

« **PREVIOUS**
Hibernate Tomcat JNDI DataSource Example Tutorial

NEXT »
Hibernate One To Many Mapping Example Annotation

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: HIBERNATE

Comments

Bipin Sah says
MARCH 16, 2018 AT 12:18 PM

failed.org.hibernate.MappingException: Could not determine type for:
com.journaldev.hibernate.model.Customer, at table: TRANSACTION, for columns:
[org.hibernate.mapping.Column(customer)]
Following this code this error occurs.
Reply

raghave says
DECEMBER 12, 2016 AT 5:56 AM
Organization of content is good, but looks like there is no adequate reasoning provided. Its like "rattafication" based content that has been spilled over.
Reply

Raisuddin says
OCTOBER 3, 2016 AT 6:20 AM
Hi
i have 3 table and want to map like… one to many between 1 and 2 table and one to one in 2 and 3 table, plz help to mapping these
Reply

janardhan says
JULY 6, 2016 AT 10:09 AM
Hi this is janardhan i have small query
in hibernte one to one relational mapping i need a example such that
one to one relation with bidirectional with foreign key constraint using xml
Reply

Mayank Singh says
JULY 1, 2016 AT 9:59 PM
In above example you are using ServiceRegistry Interface for creating the SessionFactory object. What is the advantage for using ServiceRegistry .
We can also directly use
SessionFactory sf = configuration.buildSessionFactory();
Reply

Sergey says
MAY 26, 2016 AT 12:01 PM
Hello, thank you for your tutorials it is very useful for beginners. I think there is a little mistake in your ERD. In MySql tables you have defined "txn_id" in "Customer" table as FK for "Transaction" table. That mean ERD should be like "Customer -> Transaction" but you have "Transaction -> Customer".
P.S. Sorry for my English it's not my native language.
Reply

Supratim Basu Roy says
JULY 30, 2015 AT 1:51 PM
using your above JPA version – I am getting the below exception
Exception occured. null id generated for:class com.springhib.domain.Customer
org.hibernate.id.IdentifierGenerationException: null id generated for:class
Reply

Ram Sharan says
JULY 23, 2018 AT 1:35 PM
Use @IdClass(Customer.class) at Customer entity level.
Reply

Gouravmoy Mohanty says
JUNE 6, 2015 AT 1:59 AM
how to fetch the mapped object? When i am trying, the mapped object like the customer object in txn is showing null. Both with eagar and lazy fetch type.
Reply

jyoti says
FEBRUARY 25, 2015 AT 10:58 AM
Excellent explanation…
thank you.
As vikash said please post purpose and meaning of these annotations as well.
Reply

Vikash says
NOVEMBER 17, 2014 AT 5:56 PM
It would have been better, If you have explained the purpose and meaning of these anotations.
Reply

Edgar says
SEPTEMBER 11, 2014 AT 8:27 AM
Very well explained. But,
How can I insert only in the table "Txn" with a existent "Id Customer"?
Thanx
Reply

sahej says
JULY 14, 2014 AT 1:14 AM
i want to understand what does "name = property" means in case of foriegn…..
why is the name attribute refering to property…?
As in case of sequence i do understand
what name = sequence means….
Reply

Peter Jerald says
MAY 11, 2014 AT 10:44 AM
Very well explained . Thanks .
Reply

sahej says
JULY 14, 2014 AT 1:11 AM
i want to understand what does txn means in case of foriegn…..
why is the name attribute refering to property…?
As in case of sequence i do understand
name of the sequence created in oracle db
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

HIBERNATE FRAMEWORK

Hibernate Tutorial

- Hibernate Example
- Hibernate SessionFactory
- Hibernate Session get load
- Hibernate Session save
- HQL Example
- Hibernate Criteria
- Hibernate SQL
- Hibernate Named Query
- Hibernate Log4J
- Hibernate Validator
- Hibernate Tomcat DataSource

Hibernate Mapping

- Hibernate One to One Mapping
- Hibernate One to Many Mapping
- Hibernate Many to Many Join Tables

Hibernate Caching

- Hibernate Cache
- Hibernate EHCache

Hibernate Integrations

- Hibernate Spring
- Hibernate Spring MVC
- Hibernate Struts 2
- Hibernate Primefaces
- Hibernate Primefaces Spring
- Hibernate SpringRoo Primefaces
- Hibernate JSF Spring

Miscellaneous

- Hibernate Tools Eclipse Plugin
- Hibernate Configuration Offline
- [Solved] No identifier specified
- Hibernate Program Not Terminating
- Access to DialectResolutionInfo
- get is not valid
- No CurrentSessionContext configured
- Hibernate Interview Questions

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial