Hibernate 3 introduction and writing hello world application

NOVEMBER 12, 2012 LOKESH 24 COMMENTS

Hibernate was started in 2001 by Gavin King as an alternative to using EJB2-style entity beans. Its mission back then was to simply offer better persistence capabilities than offered by EJB2 by simplifying the complexities and allowing for missing features. Hibernate used its mapping files and configuration files to achieve its objectives. With the introduction of **annotations** in java community with JDK 1.5, Hibernate community started working on Hibernate 3, which has support for annotations.

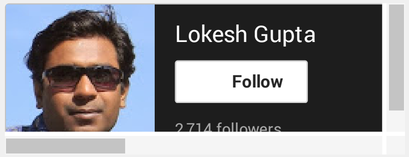
Download Source code

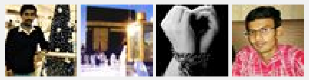
In this post, I will try to detail out more information on hibernate and then will identify the basic steps to use hibernate for our first running example application.

Sections in this post:

What is hibernate
How hibernate works
Hibernate architecture
Relation with JPA
Writing our first application

What is hibernate





Hibernate is an open source object/relational mapping tool for Java. It provides a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities and can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC.

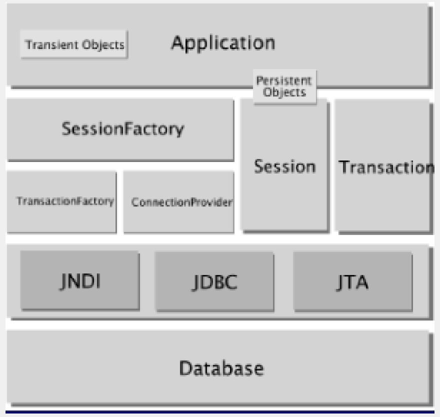
Mapping Java classes to database tables is accomplished through the configuration of an XML file or by using Java Annotations. Facilities to arrange one-to-many and many-to-many relationships between classes are provided. In addition to managing associations between objects, Hibernate can also manage reflexive associations where an object has a one-to-many relationship with other instances of its own type.

How hibernate works

Hibernate doesn't get in your way; nor does it force you to change the way your objects behave. They don't need to implement any magical interfaces in order to be blessed with the ability to persist. **All you have to do to put some annotations telling hibernate that how to use them when mapping them with database**. At runtime, hibernate reads these annotations and use this information to build queries to send to some relational database.

There is a simple, intuitive API in Hibernate to perform queries against the objects represented by the database. **To change those objects you just interact with them normally in the program, and then tell Hibernate to save the changes**. Creating new objects is similarly simple; you just create them in the normal way and tell Hibernate about them using annotations so they can get stored to the database.

Hibernate architecture



The above diagram shows a **comprehensive architecture** of Hibernate. Here are some definitions of the objects depicted in the diagrams:

Application:
The main application consist of all user written java files and other resources.

Transient Objects:
Instances of persistent classes that are not currently associated with a Session. They may have been instantiated by the application and not yet persisted, or they may have been instantiated by a closed Session.

Persistent objects:
Short-lived, single threaded objects containing persistent state and business function. These can be ordinary Java Beans/POJOs. They are associated with exactly one Session. Once the Session is closed, they will be detached and free to use in any application layer (for example, directly as data transfer objects to and from presentation).

SessionFactory:
A thread safe, immutable cache of compiled mappings for a single database. A factory for Session and a client of ConnectionProvider, SessionFactory can hold an optional (second-level) cache of data that is reusable between transactions at a process, or cluster, level.

Session:
A single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps a JDBC connection and is a factory for Transaction. Session holds a mandatory first-level cache of persistent objects that are used when navigating the object graph or looking up objects by identifier.

TransactionFactory:
(Optional) A factory for Transaction instances. It is not exposed to the application, but it can be extended and/or implemented by the developer.

ConnectionProvider:
(Optional) A factory for, and pool of, JDBC connections. It abstracts the application from underlying Data source or DriverManager. It is not exposed to application, but it can be extended and/or implemented by the developer.

Transaction:
(Optional) A single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC, JTA or CORBA transaction. A Session might span several Transactions in some cases. However, transaction demarcation, either using the underlying API or Transaction, is never optional.

Relation with JPA

JPA (Java Persistence API) is an interface for persistence providers to implement. Hibernate is one such implementation of JPA. You can annotate your classes as much as you would like with JPA annotations, however without an implementation nothing will happen. **Think of JPA as the guidelines that must be followed or an interface, while Hibernates JPA implementation is code that meets the API as defined by JPA and provides the under the hood functionality**.

When you use hibernate with JPA you are actually using the Hibernate JPA implementation. The benefit of this is that you can swap out hibernates implementation of JPA for another implementation of the JPA specification. When you use straight hibernate your locking into the implementation because other ORMs may use different methods/configurations and annotations, therefore you cannot just switch over to another ORM.

Writing our first application

So, we have got a good overview of : what hibernate is all about. Now, its time to do some practical work. Lets create out first Hello world application. In this application, I have created an Employee class and declared four attributes.

- id
- email
- firstname and
- lastname

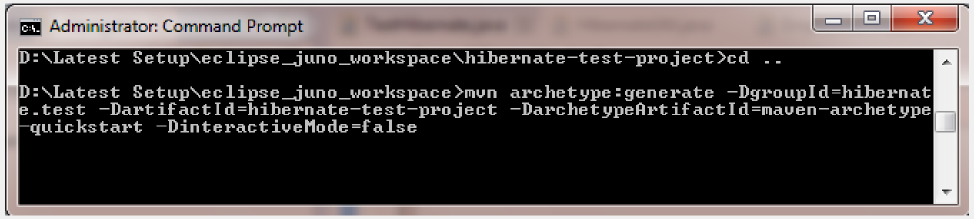
I want the id attribute should be generated automatically so that application code does not store a local cahche of employee ids.

So far we targeted what we want to make in our first application. Lets identify the files need to be created.

1) hibernate.cfg.xml // This configuration file will be used to store database connection information and schema level settings.
2) EmployeeEntity.java //This class will be java POJO having hibernate annotations.
3) HibernateUtil.java //This class will have utility methods which will be used for creating session factory and session objects.
4) TestHibernate.java //This class will be used to test our configuration settings and Emplyee entity annotations.

Before moving into code, lets see the project setup and adding **maven** dependencies which need to added to pom.xml to include all compile time and runtime dependencies.

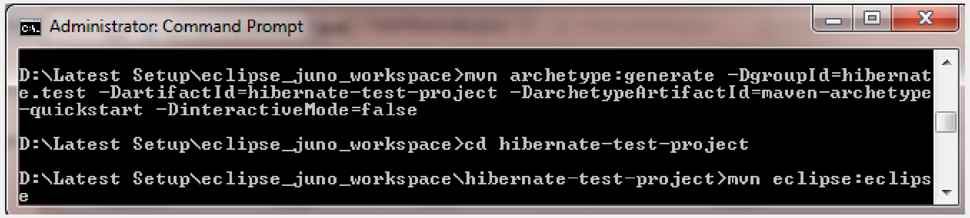
A) Create a maven project



```
D:\Latest Setup\eclipse_juno_workspace\hibernate-test-project>cd ..

D:\Latest Setup\eclipse_juno_workspace>mvn archetype:generate -DgroupId=hibernat
e.test -DartifactId=hibernate-test-project -DarchetypeArtifactId=maven-archetype
-quickstart -DinteractiveMode=false
```

2) Make project to support eclipse

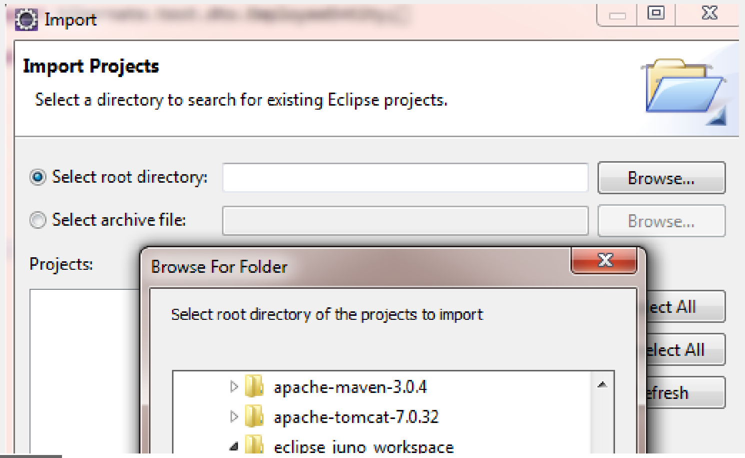


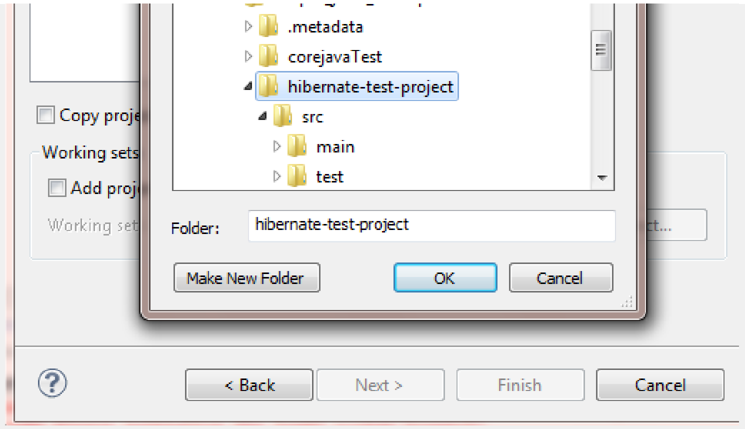
```
D:\Latest Setup\eclipse_juno_workspace>mvn archetype:generate -DgroupId=hibernat
e.test -DartifactId=hibernate-test-project -DarchetypeArtifactId=maven-archetype
-quickstart -DinteractiveMode=false

D:\Latest Setup\eclipse_juno_workspace>cd hibernate-test-project

D:\Latest Setup\eclipse_juno_workspace\hibernate-test-project>mvn eclipse:eclips
e
```

3) Now import java project to eclipse workspace.




.metadata
corejavaTest
hibernate-test-project
src
main
test
Folder: hibernate-test-project

Above steps will create the minimum setup. Now its time to add hibernate dependencies.

Maven dependencies in pom.xml

```
<dependency>
    <groupid>org.hibernate</groupid>
    <artifactid>hibernate-commons-annotations</artifactid>
    <version>3.0.0.ga</version>
</dependency>
<dependency>
    <groupid>org.hibernate</groupid>
    <artifactid>hibernate-annotations</artifactid>
    <version>3.3.0.ga</version>
</dependency>
<dependency>
    <groupid>mysql</groupid>
    <artifactid>mysql-connector-java</artifactid>
    <version>5.1.6</version>
</dependency>
<dependency>
    <groupid>antlr</groupid>
    <artifactid>antlr</artifactid>
    <version>2.7.6</version>
</dependency>
<dependency>
    <groupid>commons-collections</groupid>
    <artifactid>commons-collections</artifactid>
    <version>3.1</version>
</dependency>
<dependency>
    <groupid>dom4j</groupid>
    <artifactid>dom4j</artifactid>
    <version>1.6.1</version>
</dependency>
<dependency>
    <groupid>javassist</groupid>
    <artifactid>javassist</artifactid>
    <version>3.4.GA</version>
</dependency>
<dependency>
    <groupid>javax.transaction</groupid>
    <artifactid>jta</artifactid>
    <version>1.1</version>
</dependency>
<dependency>
    <groupid>org.slf4j</groupid>
    <artifactid>slf4j-api</artifactid>
    <version>1.5.6</version>
</dependency>
<dependency>
    <groupid>org.slf4j</groupid>
    <artifactid>slf4j-log4j12</artifactid>
    <version>1.5.6</version>
</dependency>
```

Please note that we are not using all dependencies in this sample program but they will be used when we start expanding our application.

Adding hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatetest</property>
        <property name="hibernate.connection.password">lg225295</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <mapping class="hibernate.test.dto.EmployeeEntity"></mapping>
    </session-factory>
</hibernate-configuration>
```

Do not forget to set correct password before running the application.

Adding EmployeeEntity.java

```
package hibernate.test.dto;

import java.io.Serializable;

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;

import org.hibernate.annotations.OptimisticLockType;

@Entity
@org.hibernate.annotations.Entity(optimisticLock = OptimisticLockType.ALL)
@Table(name = "Employee", uniqueConstraints = {
        @UniqueConstraint(columnNames = "ID"),
        @UniqueConstraint(columnNames = "EMAIL") })
public class EmployeeEntity implements Serializable {

    private static final long serialVersionUID = -1798070786993154676L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", unique = true, nullable = false)
    private Integer employeeId;

    @Column(name = "EMAIL", unique = true, nullable = false, length = 100)
    private String email;

    @Column(name = "FIRST_NAME", unique = false, nullable = false, length = 100)
    private String firstName;

    @Column(name = "LAST_NAME", unique = false, nullable = false, length = 100)
    private String lastName;

    // Accessors and mutators for all four fields
}
```

Adding HibernateUtil.java

```
package hibernate.test;

import java.io.File;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            return new AnnotationConfiguration().configure(
                    new File("hibernate.cgf.xml")).buildSessionFactory();

        } catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public static void shutdown() {
        // Close caches and connection pools
        getSessionFactory().close();
    }
}
```

Please correct the path of hibernate.cgf.xml.

Testing our code

```
package hibernate.test;

import hibernate.test.dto.EmployeeEntity;
import org.hibernate.Session;

public class TestHibernate {

    public static void main(String[] args) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();

        // Add new Employee object
        EmployeeEntity emp = new EmployeeEntity();
        emp.setEmail("demo-user@mail.com");
        emp.setFirstName("demo");
        emp.setLastName("user");

        session.save(emp);

        session.getTransaction().commit();
        HibernateUtil.shutdown();
    }
}
```

Above code will create a new table employee in database and insert one row in this table. In logs you can verify the insert statement which got executed.

```
Hibernate: insert into Employee (EMAIL, FIRST_NAME, LAST_NAME) values (?, ?, ?)
```

If you face problem in running above application, drop me a comment and i will be glad to discuss the problem with you.

Happy Learning!!

Download Source code

Related Posts:

1. **Hibernate insert query tutorial**
2. **Hibernate EhCache configuration tutorial**
3. **Spring 3 mvc hello world application with maven and jstl**
4. **RESTEasy + Tomcat hello world application**
5. **Hibernate example of insert/select blob from database**
6. **Hibernate OSCache configuration example tutorial**

HELLO WORLD APPLICATION · HIBERNATE · HIBERNATE ARCHITECTURE · MAVEN DEPENDANCY FOR HIBERNATE · RELATIONAL MAPPING TOOL

PREVIOUS POST
Revisiting memory management and garbage collection mechanisms in java

NEXT POST
Hibernate one-to-one mapping using annotations

24 THOUGHTS ON “HIBERNATE 3 INTRODUCTION AND WRITING HELLO WORLD APPLICATION”

dilhar

OCTOBER 27, 2014 AT 6:20 AM

good start..but its better could you explain the annotations using inside the entity as someone may be new to hibernate

REPLY

★ Lokesh

OCTOBER 27, 2014 AT 6:28 AM

You may want to read this: http://howtodoinjava.com/2014/10/21/hibernate-jpa-2-persistence-annotations-tutorial/

REPLY

dilhar

OCTOBER 27, 2014 AT 12:06 PM

thanks.

REPLY

Ravikanth

OCTOBER 15, 2014 AT 6:46 PM

Hi Lokesh,

I have basic question regarding when, why we should use hibernate/JDBC Spring(DI/AOP)/CoreJava?
When(and why) we should choose hibernate or jdbc or other ORM? what are the advantages of one over the other?
If I have a application with some 20 tables and I want to build some application using those 20 tables? would you suggest Hibernate?
What technology would you choose – Spring, Hibernate or Just coreJava/Hibernate or coreJava/JDBC or Spring/JDBC and why?

I know there are lot of resources/discussion available online, but I would like to know from you. Please suggest.

ThankYou,
Ravikanth

REPLY

★ Lokesh

OCTOBER 16, 2014 AT 3:19 AM

Hi Ravi, Its very good (i.e. difficult) question to answer. But let me put my thoughts at this very moment (a good debate might change them and I am open for it).
1) If you working on a small utility type project (e.g. data importer or exporter). I will recommend to use JDBC directly as it provides good control as well as performance.
2) If you are working on a project which will be used by hundreds/thousands of users; then i will suggest to use hibernate (not so popular advise). Though all below features you can achieve with JBDC as well, but **cost would be longer development time and possibly buggy code** as frameworks have been refined with help of thousands community members over years.

– Automatic Transaction Support
– Inbuilt caching capabilities
– DB schema non-dependent code

REPLY

HIMANSU NAYAK

JULY 6, 2014 AT 1:06 PM

Hi Lokesh,

If possible upgrade the code to Hibernate 4 compatible as many api for createing Session Factory are deprecated.

REPLY

★ Lokesh

JULY 6, 2014 AT 4:42 PM

Well suggested. I will work on this.

REPLY

Amutha

MARCH 29, 2014 AT 11:09 AM

Hi Lokesh,

Awesome work!!! Too impressive!!!

The way u r explaining is not complicating the things.

So pls describe as simple as possible because simple things reach a lot.

REPLY

Puneet

JANUARY 24, 2014 AT 7:33 PM

Hi Lokesh.. I have a question…… there are other orm tools like iBatis , toplink etc available in the java framework.. Why Hibernate is most demanding & generally used?

REPLY

★ Lokesh

JANUARY 25, 2014 AT 3:11 AM

Puneet, I have worked on hibernate and iBatis both. Based on my experience I would say that if your domain objects are similar to DB tables/columns and you do not need any complex SQL control then go for hibernate. Else choose iBatis. iBatis really gives you great control over complex database queries. It gives you almost all features (e.g. caching) which hibernate provides.
But here is a catch. iBatis demands extra code/config (sql-maps i.e. mapping between java objects and database columns), which hibernate takes care for you.

Development time is considerably short in Hibernate in comparison to iBatis, but you loose the flexibility which iBatis provide.

Note: I have compared hibernate and iBatis based on my experience. Other frameworks I really can not comment on without further study.

REPLY

nouranMhmoud

DECEMBER 29, 2013 AT 6:50 PM

what's the Libraries you've used for this tutorial? why not attached with the source code ? if it is mention it for me please . thanks.

REPLY

★ **Lokesh Gupta**

DECEMBER 29, 2013 AT 7:53 PM

Source code download link available in start and end of post. Please refer .classpath file for referenced jar files:

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
<classpathentry kind="src" path="src/test/java" output="target/test-classes" including="**/*.java"/>
<classpathentry kind="src" path="src/main/java" including="**/*.java"/>
<classpathentry kind="output" path="target/classes"/>
<classpathentry kind="var" path="M2_REPO/javax/transaction/jta/1.1/jta-1.1.jar"/>
<classpathentry kind="var" path="M2_REPO/javax/persistence/persistence-api/1.0/persistence-api-1.0.jar"/>
<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
<classpathentry kind="var" path="M2_REPO/junit/junit/3.8.1/junit-3.8.1.jar"/>
<classpathentry kind="var" path="M2_REPO/antlr/antlr/2.7.6/antlr-2.7.6.jar"/>
<classpathentry kind="var" path="M2_REPO/commons-collections/commons-collections/3.1/commons-collections-3.1.jar"/>
<classpathentry kind="var" path="M2_REPO/dom4j/dom4j/1.6.1/dom4j-1.6.1.jar"/>
<classpathentry kind="var" path="M2_REPO/xml-apis/xml-apis/1.0.b2/xml-apis-1.0.b2.jar"/>
<classpathentry kind="var" path="M2_REPO/javassist/javassist/3.4.GA/javassist-3.4.GA.jar"/>
<classpathentry kind="var" path="M2_REPO/org/slf4j/slf4j-api/1.5.6/slf4j-api-1.5.6.jar"/>
<classpathentry kind="var" path="M2_REPO/org/slf4j/slf4j-log4j12/1.5.6/slf4j-log4j12-1.5.6.jar"/>
<classpathentry kind="var" path="M2_REPO/log4j/log4j/1.2.14/log4j-1.2.14.jar"/>
<classpathentry kind="var" path="M2_REPO/org/hibernate/hibernate-commons-annotations/3.0.0.ga/hibernate-commons-annotations-3.0.0.ga.jar"/>
<classpathentry kind="var" path="M2_REPO/commons-logging/commons-logging/1.0.4/commons-logging-1.0.4.jar"/>
<classpathentry kind="var" path="M2_REPO/org/hibernate/hibernate-annotations/3.3.0.ga/hibernate-annotations-3.3.0.ga.jar"/>
<classpathentry kind="var" path="M2_REPO/org/hibernate/hibernate/3.2.1.ga/hibernate-3.2.1.ga.jar"/>
<classpathentry kind="var" path="M2_REPO/net/sf/ehcache/ehcache/1.2.3/ehcache-1.2.3.jar"/>
```

```
<classpathentry kind="var" path="M2_REPO/asm/asm-attrs/1.5.3/asm-attrs-1.5.3.jar"/>
<classpathentry kind="var" path="M2_REPO/cglib/cglib/2.1_3/cglib-2.1_3.jar"/>
<classpathentry kind="var" path="M2_REPO/asm/asm/1.5.3/asm-1.5.3.jar"/>
<classpathentry kind="var" path="M2_REPO/mysql/mysql-connector-java/5.1.6/mysql-connector-java-5.1.6.jar"/>
</classpath>
```

REPLY

nouranMhmoud

DECEMBER 29, 2013 AT 8:28 PM

thanks for the prompt reply, but I've an error I don't know why .

I downloaded the sourcecode and then I extracted the .rar project and took the src folder and pom.xml and hibernate.cgf.xml files and created with them a new project with Netbeans IDE .

after that I added the libraries required using properties>libraries>classthpath from the installation of hibernate framework and junit framework.

then this error appeared .

```
Initial SessionFactory creation failed.java.lang.IncompatibleClassChangeError: Implementing class
Exception in thread "main" java.lang.ExceptionInInitializerError
at hibernate.test.HibernateUtil.buildSessionFactory(HibernateUtil.java:20)
at hibernate.test.HibernateUtil.(HibernateUtil.java:9)
at hibernate.test.TestHibernate.main(TestHibernate.java:10)
Caused by: java.lang.IncompatibleClassChangeError: Implementing class
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:800)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
at java.net.URLClassLoader.defineClass(URLClassLoader.java:449)
at java.net.URLClassLoader.access$100(URLClassLoader.java:71)
at java.net.URLClassLoader$1.run(URLClassLoader.java:361)
at java.net.URLClassLoader$1.run(URLClassLoader.java:355)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(URLClassLoader.java:354)
at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:308)
at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
at hibernate.test.HibernateUtil.buildSessionFactory(HibernateUtil.java:14)
… 2 more
```

I know I didn't follow this tutorial literally but this because I've a little bit fuzzy about using hibernate framework.

hope that you can assist me and sorry for the long reply .
thanks in advance .

REPLY

★ **Lokesh Gupta**

DECEMBER 31, 2013 AT 7:10 PM

It’s definitely related to jar versions of cglib and it’s related dependencies.
Refer: http://howtodoinjava.com/2013/05/25/solved-java-lang-incompatibleclasschangeerror-implementing-class/

REPLY

venki

OCTOBER 21, 2013 AT 10:39 AM

how to add hibernate dependencies in pom.xml

REPLY

★ **Lokesh Gupta**

OCTOBER 21, 2013 AT 11:36 AM

Please refer in pom.xml file in downloaded sourcecode.

REPLY

Jay

OCTOBER 4, 2013 AT 9:58 AM

Awesome tutorial. Very helpful. Thanks for the efforts. Keep it coming 😃

REPLY

satish

AUGUST 8, 2013 AT 4:30 PM

Hi Lokesh gupta,
i am learning alot from your tutorials.your tutorials awesome.I have a small doubt what is the purpose of this annotation?
@org.hibernate.annotations.Entity(optimisticLock = OptimisticLockType.ALL)

REPLY

guillaume

DECEMBER 17, 2012 AT 7:53 AM

Hello,
thanks for the tutorial.
I had to correct two things to make it work :

– in hibernate.cfg.xml, you have to specify the mapping class, eg :

– in the entity bean, you have to specify dynamicUpdate=true into the annotation Entity :
@org.hibernate.annotations.Entity(optimisticLock = OptimisticLockType.ALL, dynamicUpdate = true)

REPLY

Lokesh Gupta

DECEMBER 17, 2012 AT 8:10 AM

Thanks for pointing out. First is definitely a typo so I have corrected this in post itself. Regarding second, I am not sure. It worked for me without dynamic update.

Anyways, as you have logged your observation here, it will definitely help someone if faced problem.

REPLY

venkata

MARCH 16, 2014 AT 8:42 PM

hi, can you explain me clearly what does this mean

@org.hibernate.annotations.Entity(optimisticLock = OptimisticLockType.ALL, dynamicUpdate = true)

REPLY

★ Lokesh

MARCH 17, 2014 AT 6:38 PM

1) Optimistic means….the table is open for read/write over entire network for all users/sessions. We can move the cursor, backward or forward dynamically.
2) Pessimistic means… the table is open for read/write only for that current session. The other session users can not edit the same.

If set the dynamic-update to true, which means exclude unmodified properties in the Hibernate's SQL update statement. It means, that if in any transaction in a table, 10 columns are present and values got updated for 3 columns, then update statement created by hibernate will have only 3 column names.

REPLY

Ginu

MARCH 20, 2014 AT 7:16 AM

Yes dynamic update property had to be added.
The only difference being I connected to postgresql

REPLY

Pingback: Hibernate one-to-one mapping using annotations « How to do in "JAVA"

Note:- In comment box, please put your code inside **[java] ... [/java]** OR **[xml] ... [/xml]** tags otherwise it may not appear as intended.

LEAVE A REPLY

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

Name *

Email *

Website

Are you Human and NOT robot? Solve this to prove !! *

+ 1 = three

Comment

You may use these HTML tags and attributes:

```
<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>
```

POST COMMENT

☐ Notify me of follow-up comments by email.

☑ Notify me of new posts by email.

SUBSCRIBE TO BLOG VIA EMAIL

Join 3,678 other subscribers

Email Address

Email Address

SUBSCRIBE

REFERENCES

Oracle Java Docs
Spring 3 Reference
Spring 4 References
Jboss RESTEasy JAX-RS
Hibernate Developer Guide
Junit Wiki
Maven FAQs
Dzone Dev Links
JSP Homepage
ANT Homepage

META LINKS

Advertise
Share your knowledge
Privacy policy
Contact Us
About Us

POPULAR POSTS

Spring 3 and hibernate integration tutorial with example
Reading/writing excel files in java : POI tutorial
How hashmap works in java
Singleton design pattern in java
Useful java collection interview questions

Understanding hibernate first level cache with example

How hibernate second level cache works?

How to make a java class immutable

Working with hashCode and equals methods in java

4 ways to schedule tasks in Spring 3 : @Scheduled example