|  |  |
| --- | --- |
|  | **Use Of XML, Annotations and Property Configuration** |
|  |  |
|  | **Hibernate** is one of the most widely used **Java ORM** tool. |
|  | **Object-relational mapping** or **ORM** is the programming technique to map application domain model objects to the relational database tables. Hibernate is java based ORM tool that provides framework for mapping application domain objects to the relational database tables and vice versa. |
|  | Some of the benefits of using Hibernate as ORM tool are: |
| 1. | Hibernate supports mapping of java classes to database tables and vice versa. It provides features to perform CRUD operations across all the major relational databases. |
| 2. | Hibernate eliminates all the boiler-plate code that comes with JDBC and takes care of managing resources, so we can focus on business use cases rather than making sure that database operations are not causing resource leaks. |
| 3. | Hibernate supports transaction management and make sure there is no inconsistent data present in the system. |
| 4. | Since we use XML, property files or annotations for mapping java classes to database tables, it provides an abstraction layer between application and database. |
| 5. | Hibernate helps us in mapping joins, collections, inheritance objects and we can easily visualize how our model classes are representing database tables. |
| 6. | Hibernate provides a powerful query language (HQL) that is similar to SQL. However, HQL is fully object-oriented and understands concepts like inheritance, polymorphism and association. |
| 7. | Hibernate is an open source project from Red Hat Community and used worldwide. This makes it a better choice than others because learning curve is small. |
| 8. | Hibernate is easy to integrate with other Java EE frameworks, it’s so popular that [Spring Framework](http://www.journaldev.com/2593/spring-jdbc-example) provides built-in support for integrating hibernate with Spring applications. |
| 9. | Hibernate also offers integration with some external modules. For example Hibernate Validator is the reference implementation of Bean Validation |
|  |  |
|  | Hibernate Architecture |
|  | 1.png |
| 1. | **SessionFactory (org.hibernate.SessionFactory)**: |
|  | SessionFactory is an immutable thread-safe cache of compiled mappings for a single database. We can get instance of org.hibernate.Session using SessionFactory. |
| 2. | **Session (org.hibernate.Session)**: |
|  | Session is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps JDBC java.sql.Connection and works as a factory for org.hibernate.Transaction. |
| 3. | **Persistent objects**: |
|  | Persistent objects are short-lived, single threaded objects that contains persistent state and business function. These can be ordinary JavaBeans/POJOs. They are associated with exactly one org.hibernate.Session. |
| 4. | **Transient objects**: |
|  | Transient objects are persistent classes instances that are not currently associated with a org.hibernate.Session. They may have been instantiated by the application and not yet persisted, or they may have been instantiated by a closed org.hibernate.Session. |
| 5. | **Transaction (org.hibernate.Transaction)**: |
|  | Transaction is a single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC or JTA transaction. A org.hibernate.Session might span multiple org.hibernate.Transaction in some cases. |
| 6. | **ConnectionProvider (org.hibernate.connection.ConnectionProvider)**: |
|  | ConnectionProvider is a factory for JDBC connections. It provides abstraction between the application and underlying javax.sql.DataSource or java.sql.DriverManager. It is not exposed to application, but it can be extended by the developer. |
| 7. | **TransactionFactory (org.hibernate.TransactionFactory)**: |
|  | A factory for org.hibernate.Transaction instances. |
|  | Hibernate Example |
|  | When developing hibernate applications, we need to provide two set of configuration. |
|  | First set of configuration contains database specific properties that will be used to create Database connection and Session objects. |
|  | Second set of configurations contains mapping between model classes and database tables. |
|  | We can use XML based or properties based configuration for database connection related configurations. We can use XML based or annotation based configurations for providing model classes and database tables mapping. |
|  | We will use JPA annotations from javax.persistence for annotation based mappings. |
|  | Our final project will look like below image. |
|  | 1.png |
|  | 2.png |
|  |  |
|  |  |
|  |  |
|  | Database Table Setup |
|  | For my example, I am using MySQL database and below script is used to create necessary table. |
|  | 3.png |
|  | Hibernate Project Dependencies |
|  | Our final pom.xml file looks like below. |
|  | 4.png |
|  | 5.png |
|  | 6.png |
|  | **hibernate-core** artifact contains all the core hibernate classes, so we will get all the necessary features by including it in the project. |
|  | Hibernate 4 uses JBoss logging but older versions uses slf4j for logging purposes, so I have included **slf4j-simple** artifact in my project, although not needed because I am using Hibernate 4. |
|  | **mysql-connector-java** is the MySQL driver for connecting to MySQL databases, if you are using any other database then add corresponding driver artifact. |
|  | Domain Model Classes |
|  | As you can see in above image that we have two model classes, Employee and Employee1. |
|  | Employee is a simple Java Bean class and we will use XML based configuration for providing it’s mapping details. |
|  | Employee1 is a java bean where fields are annotated with JPA annotations, so that we don’t need to provide mapping in separate XML file. |
|  | 12.png |
|  | 8.png |
|  | 9.png |
|  | 10.png |
|  | 11.png |
|  | Employee class is simple java bean, there is nothing specific to discuss here. |
|  | 13.png |
|  | 14.png |
|  | 16.png |
|  | 17.png |
|  | 18.png |
|  | 19.png |
|  | javax.persistence.Entity annotation is used to mark a class as Entity bean that can be persisted by hibernate, since hibernate provides JPA implementation. |
|  | javax.persistence.Table annotation is used to define the table mapping and unique constraints for the columns. |
|  | javax.persistence.Id annotation is used to define the primary key for the table. javax.persistence.GeneratedValue is used to define that the field will be auto generated and **GenerationType.IDENTITY** strategy is used so that the generated “id” value is mapped to the bean and can be retrieved in the java program. |
|  | javax.persistence.Column is used to map the field with table column, we can also specify length, nullable and uniqueness for the bean properties. |
|  | Hibernate Mapping XML Configuration |
|  | we will use XML based configuration for Employee class mapping. We can choose any name, but it’s good to choose with table or java bean name for clarity. Our hibernate mapping file for Employee bean looks like below. |
|  | **employee.hbm.xml** |
|  | 20.png |
|  | 21.png |
|  | The xml configuration is simple and does the same thing as annotation based configuration. |
|  | Hibernate Configuration Files |
|  | We will create two hibernate configuration xml files – one for xml based configuration and another for annotation based configuration. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **hibernate.cfg.xml** |
|  | 22.png |
|  | 23.png |
|  | 24.png |
|  | 25.png |
|  | name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property> |
|  | 26.png |
|  | Most of the properties are related to database configurations, other properties details are given in comment. Note the configuration for hibernate mapping file, we can define multiple hibernate mapping files and configure them here. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | **hibernate-annotation.cfg.xml** |
|  | **27.png** |
|  | name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property> |
|  | 28.png |
|  | 29.png |
|  | name="hibernate.current\_session\_context\_class">thread</property> |
|  | 30.png |
|  | Most of the configuration is same as XML based configuration, the only difference is the mapping configuration. We can provide mapping configuration for classes as well as packages. |
|  | Hibernate SessionFactory |
|  | I have created a utility class where I am creating SessionFactory from XML based configuration as well as property based configuration. For property based configuration, we could have a property file and read it in the class, but for simplicity I am creating Properties instance in the class itself. |
|  | 31.png |
|  | org.hibernate.boot.registry.StandardServiceRegistryBuilder; |
|  | 33.png |
|  | 35.png |
|  | 36.png |
|  | 37.png |
|  | 38.png |
|  | 39.png |
|  | 40.png |
|  | StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build(); |
|  | 41.png |
|  | 42.png |
|  | 43.png |
|  | 45.png |
|  | 46.png |
|  | StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build(); |
|  | 47.png |
|  | 48.png |
|  | 49.png |
|  | 50.png |
|  | Creating SessionFactory for XML based configuration is same whether mapping is XML based or annotation based. For properties based, we need to set the properties in Configuration object and add annotation classes before creating the SessionFactory. |
|  | Overall creating SessionFactory includes following steps: |
|  | 1. Creating Configuration object and configure it |
|  | 2. Creating ServiceRegistry object and apply configuration settings. |
|  | 3. Use configuration.buildSessionFactory() by passing ServiceRegistry object as argument to get the SessionFactory object. |
|  | Our application is almost ready now, let’s write some test programs and execute them. |
|  | Hibernate XML Configuration Test |
|  | 1.png |
|  | 2.png |
|  | 3.png |
|  | The program is self understood, when we execute the test program, we get following output. |
|  | May 06, 2014 12:40:06 AM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit> |
|  | 4.png |
|  | 5.png |
|  | 6.png |
|  | 7.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure |
|  | 8.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | 9.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | 10.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | 11.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure |
|  | 12.png |
|  | org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService |
|  | 13.png |
|  | org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init> |
|  | 14.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop  INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost/TestDB] |
|  | Notice that it’s printing the generated employee id, you can check database table to confirm it. |
|  | Hibernate Annotation Configuration Test |
|  | 1.png |
|  | 2.png |
|  | 3.png |
|  | 4.png |
|  | When we execute above program, we get following output. |
|  | May 06, 2014 12:42:22 AM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit> |
|  | 5.png |
|  | 7.png |
|  | 8.png |
|  | 9.png |
|  | 10.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure |
|  | 11.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | 12.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | 13.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | 14.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure |
|  | 15.png |
|  | 16.png |
|  | org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService |
|  | 1.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop |
|  | 3.png |
|  | Have a look at the output and compare it with the output from the XML based configuration, you will notice some differences. For example, we are not setting connection pool size for annotation based configuration, so it’s setting to default value 20. |
|  | Hibernate Java Configuration Test |
|  | 4.png |
|  | 5.png |
|  | 6.png |
|  | 7.png |
|  | Output of the above test program is: |
|  | May 06, 2014 12:45:09 AM org.hibernate.annotations.common.reflection.java.JavaReflectionManager <clinit> |
|  | 8.png |
|  | 9.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure |
|  | 10.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | 11.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | 12.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator |
|  | INFO: HHH000006: Autocommit mode: false  May 06, 2014 12:45:09 AM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure |
|  | 13.png |
|  | org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService |
|  | 14.png |
|  | org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init> |
|  | 15.png |
|  | org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop |
|  | 16.png |