Hibernate Getting Started Guide

Table of Contents

Preface

Working with both Object-Oriented software and Relational Databases can be cumbersome and time consuming. Development costs are significantly higher due to a paradigm mismatch between how data is represented in objects versus relational databases. Hibernate is an Object/Relational Mapping (ORM) solution for Java environments. The term Object/Relational Mapping refers to the technique of mapping data between an object model representation to a relational data model representation. See http://en.wikipedia.org/wiki/Object-relational_mapping for a good high-level discussion. Also, Martin Fowler's OrmHate (http://martinfowler.com/bliki/OrmHate.html) article takes a look at many of the mismatch problems.

Although having a strong background in SQL is not required to use Hibernate, having a basic understanding of the concepts can help you understand Hibernate more quickly and fully. An understanding of data modeling principles is especially important. Both http://www.agiledata.org/essays/dataModeling101.html and http://en.wikipedia.org/wiki/Data_modeling are good starting points for understanding these data modeling principles.

Hibernate takes care of the mapping from Java classes to database tables, and from Java data types to SQL data types. In addition, it provides data query and retrieval facilities. It can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC. Hibernate's design goal is to relieve the developer from 95% of common data persistence-related programming tasks by eliminating the need for manual, hand-crafted data processing using SQL and JDBC. However, unlike many other persistence solutions, Hibernate does not hide the power of SQL from you and guarantees that your investment in relational technology and knowledge is as valid as always.

Hibernate may not be the best solution for data-centric applications that only use stored-procedures to implement the business logic in the database, it is most useful with object-oriented domain models and business logic in the Java-based middle-tier. However, Hibernate can certainly help you to remove or encapsulate vendor-specific SQL code and streamlines the common task of translating result sets from a tabular representation to a graph of objects.

See http://hibernate.org/orm/contribute/ for information on getting involved.

> The projects and code for the tutorials referenced in this guide are available as hibernate-tutorials.zip

1. Obtaining Hibernate

1.1. The Hibernate Modules/Artifacts

Hibernate's functionality is split into a number of modules/artifacts meant to isolate dependencies (modularity).

hibernate-core

The main (core) Hibernate module. Defines its ORM features and APIs as well as the various integration SPIs.

hibernate-entitymanager

Defines Hibernate's support for JPA (http://jcp.org/en/jsr/detail?id=317)

hibernate-java8

Support for using Java8 specific data-types such as any of the JSR 310 Date/Time types in domain model.

hibernate-envers

Hibernate's historical entity versioning feature

hibernate-spatial

Hibernate's Spatial/GIS data-type support

hibernate-osgi

Hibernate support for running in OSGi containers.

hibernate-c3p0

Integrates the C3P0 (http://www.mchange.com/projects/c3p0/) connection pooling library into Hibernate

hibernate-hikaricp

Integrates the HikariCP (http://brettwooldridge.github.io/HikariCP/) connection pooling library into Hibernate

hibernate-proxool

Integrates the Proxool (http://proxool.sourceforge.net/) connection pooling library into Hibernate

hibernate-jcache

Integrates the JCache (https://jcp.org/en/jsr/detail?id=107) caching specification into Hibernate, enabling any compliant implementation to become a second-level cache provider.

hibernate-ehcache

Integrates the Ehcache (http://ehcache.org/) caching library into Hibernate as a second-level cache provider.

hibernate-infinispan

Integrates the Infinispan (http://infinispan.org/) caching library into Hibernate as a second-level cache provider.

1.2. Release Bundle Downloads

The Hibernate team provides release bundles hosted on the SourceForge File Release System, in both `TGZ` and `ZIP` formats. Each release bundle contains `JAR` files, documentation, source code, and other goodness.

You can download releases of Hibernate, in your chosen format, from the list at https://sourceforge.net/projects/hibernate/files/hibernate-orm/. The release bundle is structured as follows:

- The `lib/required/` directory contains the `hibernate-core` jar and all of its dependencies. All of these jars are required to be available on your classpath no matter which features of Hibernate are being used.
- The `/lib/jpa/` directory contains the `hibernate-entitymanager` jar as well as all of its dependencies (beyond those in `lib/required/`)
- The `lib/java8/` directory contains the `hibernate-java8` jar and all of its dependencies (beyond those in `lib/required/`)
- The `lib/envers` directory contains the `hibernate-envers` jar and all of its dependencies (beyond those in `lib/required/` and `lib/jpa/`).
- The `lib/spatial/` directory contains the `hibernate-spatial` jar and all of its dependencies (beyond those in `lib/required/`)
- The `lib/osgi/` directory contains the `hibernate-osgi` jar and all of its dependencies (beyond those in `lib/required/` and `lib/jpa/`)
- The `lib/optional/` directory contains the jars needed for the various connection pooling and second-level cache integrations provided by Hibernate, along with their dependencies.

1.3. Maven Repository Artifacts

The authoritative repository for Hibernate artifacts is the JBoss Maven repository. The Hibernate artifacts are synced to Maven Central as part of an automated job (some small delay may occur).

The team responsible for the JBoss Maven repository maintains a number of Wiki pages that contain important information:

- http://community.jboss.org/docs/DOC-14900 - General information about the repository.
- http://community.jboss.org/docs/DOC-15170 - Information about setting up the JBoss repositories in order to do development work on JBoss projects themselves.
- http://community.jboss.org/docs/DOC-15169 - Information about setting up access to the repository to use JBoss projects as part of your own software.

The Hibernate ORM artifacts are published under the `org.hibernate` groupId.

2. Tutorial Using Native Hibernate APIs and hbm.xml Mapping

> This tutorial is located within the download bundle under `basic/` .

Objectives

☑ Bootstrap a Hibernate `SessionFactory`

☑ Use Hibernate mapping (`hbm.xml`) files to provide mapping information

☑ Use the Hibernate native APIs

2.1. The Hibernate configuration file

For this tutorial, the `hibernate.cfg.xml` file defines the Hibernate configuration information.

The `connection.driver_class` , `connection.url` , `connection.username` and `connection.password` `<property/>` elements define JDBC connection information. These tutorials utilize the H2 in-memory database, so the values of these properties are all specific to running H2 in its in-memory mode. `connection.pool_size` is used to configure the number of connections in Hibernate's built-in connection pool.

> The built-in Hibernate connection pool is in no way intended for production use.

It lacks several features found on production-ready connection pools.

The `dialect` property specifies the particular SQL variant with which Hibernate will converse.

In most cases, Hibernate is able to properly determine which dialect to use. This is particularly useful if your application targets multiple databases.

The `hbm2ddl.auto` property enables automatic generation of database schemas directly into the database.

Finally, add the mapping file(s) for persistent classes to the configuration. The `resource` attribute of the `<mapping/>` element causes Hibernate to attempt to locate that mapping as a classpath resource using a `java.lang.ClassLoader` lookup.

There are many ways and options to bootstrap a Hibernate `SessionFactory`. For additional details, see the *Native Bootstrapping* topical guide.

2.2. The entity Java class

The entity class for this tutorial is `org.hibernate.tutorial.hbm.Event`

Notes About the Entity

- This class uses standard JavaBean naming conventions for property getter and setter methods, as well as private visibility for the fields. Although this is the recommended design, it is not required.
- The no-argument constructor, which is also a JavaBean convention, is a requirement for all persistent classes. Hibernate needs to create objects for you, using Java Reflection. The constructor can be private. However, package or public visibility is required for runtime proxy generation and efficient data retrieval without bytecode instrumentation.

2.3. The mapping file

The mapping file for this tutorial is the classpath resource `org/hibernate/tutorial/hbm/Event.hbm.xml` (as discussed above).

Hibernate uses the mapping metadata to determine how to load and store objects of the persistent class. The Hibernate mapping file is one choice for providing Hibernate with this metadata.

Example 1. The class mapping element

```xml
<class name="Event" table="EVENTS">
        ...
</class>
```

Functions of the <varname>class</varname> mapping element

- The `name` attribute (combined here with the `package` attribute from the containing `<hibernate-mapping/>` element) names the FQN of the class to be defined as an entity.
- The `table` attribute names the database table which contains the data for this entity.

Instances of the `Event` class are now mapped to rows in the `EVENTS` database table.

Example 2. The id mapping element

```xml
<id name="id" column="EVENT_ID">
    ...
</id>
```

Hibernate uses the property named by the `<id/>` element to uniquely identify rows in the table.

> It is not required for the id element to map to the table's actual primary key column(s), but it is the normal convention. Tables mapped in Hibernate do not even need to define primary keys. However, it is strongly recommend that all schemas define proper referential integrity. Therefore id and primary key are used interchangeably throughout Hibernate documentation.

The `<id/>` element here names the EVENT_ID column as the primary key of the EVENTS table. It also identifies the `id` property of the `Event` class as the property containing the identifier value.

The `generator` element informs Hibernate about which strategy is used to generated primary key values for this entity. This example uses a simple incrementing count.

Example 3. The property mapping element

```xml
<property name="date" type="timestamp" column="EVENT_DATE"/>
<property name="title"/>
```

The two `<property/>` elements declare the remaining two persistent properties of the `Event` class: `date` and `title`. The `date` property mapping includes the `column` attribute, but the `title` does not. In the absence of a `column` attribute, Hibernate uses the property name as the column name. This is appropriate for `title`, but since `date` is a reserved keyword in most databases, you need to specify a non-reserved word for the column name.

The `title mapping also lacks a type attribute. The types declared and used in the mapping files are neither Java data types nor SQL database types. Instead, they are **Hibernate mapping types**, which are converters which translate between Java and SQL data types. Hibernate attempts to determine the correct conversion and mapping type autonomously if the type attribute is not specified in the mapping, by using Java reflection to determine the Java type of the declared property and using a default mapping type for that Java type.

In some cases this automatic detection might not chose the default you expect or need, as seen with the `date` property. Hibernate cannot know if the property, which is of type `java.util.Date`, should map to a SQL *DATE*, *TIME*, or *TIMESTAMP* datatype. Full date and time information is preserved by mapping the property to the *timestamp* converter, which identifies the converter class `org.hibernate.type.TimestampType`.

Hibernate determines the mapping type using reflection when the mapping files are processed. This process adds overhead in terms of time and resources. If startup performance is important, consider explicitly defining the type to use.

2.4. Example code

The `org.hibernate.tutorial.hbm.NativeApiIllustrationTest` class illustrates using the Hibernate native API.

The examples in these tutorials are presented as JUnit tests, for ease of use. One benefit of this approach is that `setUp` and `tearDown` roughly illustrate how a `org.hibernate.SessionFactory` is created at the start-up of an application and closed at the end of the application lifecycle.

Example 4. Obtaining the `org.hibernate.SessionFactory`

```java
protected void setUp() throws Exception {
        // A SessionFactory is set up once for an application!
        final StandardServiceRegistry registry = new
StandardServiceRegistryBuilder()
                        .configure() // configures settings from hibernate.cfg.xml
                        .build();
        try {
                sessionFactory = new MetadataSources( registry
).buildMetadata().buildSessionFactory();
        }
        catch (Exception e) {
                // The registry would be destroyed by the SessionFactory, but we had
trouble building the SessionFactory
                // so destroy it manually.
                StandardServiceRegistryBuilder.destroy( registry );
        }
}
```

The `setUp` method first builds a `org.hibernate.boot.registry.StandardServiceRegistry` instance which incorporates configuration information into a working set of Services for use by the SessionFactory. In this tutorial we defined all configuration information in `hibernate.cfg.xml` so there is not much interesting to see here.

Using the `StandardServiceRegistry` we create the `org.hibernate.boot.MetadataSources` which is the start point for telling Hibernate about your domain model. Again, since we defined that in `hibernate.cfg.xml` so there is not much interesting to see here.

`org.hibernate.boot.Metadata` represents the complete, partially validated view of the application domain model which the `SessionFactory` will be based on.

The final step in the bootstrap process is to build the `SessionFactory`. The `SessionFactory` is a thread-safe object that is instantiated once to serve the entire application.

The `SessionFactory` acts as a factory for `org.hibernate.Session` instances, which should be thought of as a corollary to a "unit of work".

Example 5. Saving entities

```java
Session session = sessionFactory.openSession();
session.beginTransaction();
session.save( new Event( "Our very first event!", new Date() ) );
session.save( new Event( "A follow up event", new Date() ) );
session.getTransaction().commit();
session.close();
```

`testBasicUsage()` first creates some new `Event` objects and hands them over to Hibernate for management, using the `save()` method. Hibernate now takes responsibility to perform an *INSERT* on the database for each `Event`.

Example 6. Obtaining a list of entities

```java
session = sessionFactory.openSession();
session.beginTransaction();
List result = session.createQuery( "from Event" ).list();
for ( Event event : (List<Event>) result ) {
    System.out.println( "Event (" + event.getDate() + ") : " + event.getTitle() );
}
session.getTransaction().commit();
session.close();
```

Here we see an example of the Hibernate Query Language (HQL) to load all existing `Event` objects from the database by generating the appropriate *SELECT* SQL, sending it to the database and populating `Event` objects with the result set data.

2.5. Take it further!

Practice Exercises

- ☐ Reconfigure the examples to connect to your own persistent relational database.
- ☐ Add an association to the `Event` entity to model a message thread.

3. Tutorial Using Native Hibernate APIs and Annotation Mappings

This tutorial is located within the download bundle under `annotations/`.

Objectives

- ✓ Bootstrap a Hibernate `SessionFactory`
- ✓ Use annotations to provide mapping information
- ✓ Use the Hibernate native APIs

3.1. The Hibernate configuration file

The contents are identical to The Hibernate configuration file with one important difference… The `<mapping/>` element at the very end naming the annotated entity class using the `class` attribute.

3.2. The annotated entity Java class

The entity class in this tutorial is `org.hibernate.tutorial.annotations.Event` which follows JavaBean conventions. In fact the class itself is identical to the one in The entity Java class, except that annotations are used to provide the metadata, rather than a separate mapping file.

Example 7. Identifying the class as an entity

```java
@Entity
@Table( name = "EVENTS" )
public class Event {
    ...
}
```

The `@javax.persistence.Entity` annotation is used to mark a class as an entity. It functions the same as the `<class/>` mapping element discussed in The mapping file. Additionally the `@javax.persistence.Table` annotation explicitly specifies the table name. Without this specification, the default table name would be *EVENT*.

Example 8. Identifying the identifier property

```java
@Id
@GeneratedValue(generator="increment")
@GenericGenerator(name="increment", strategy = "increment")
public Long getId() {
    return id;
}
```

`@javax.persistence.Id` marks the property which defines the entity's identifier.

`@javax.persistence.GeneratedValue` and `@org.hibernate.annotations.GenericGenerator` work in tandem to indicate that Hibernate should use Hibernate's `increment` generation strategy for this entity's identifier values.

Example 9. Identifying basic properties

```java
public String getTitle() {
    return title;
}

@Temporal(TemporalType.TIMESTAMP)
@Column(name = "EVENT_DATE")
public Date getDate() {
    return date;
}
```

As in <<hibernate-gsg-tutorial-basic-mapping, the `date` property needs special handling to account for its special naming and its SQL type.

Attributes of an entity are considered persistent by default when mapping with annotations, which is why we don't see any mapping information associated with `title`.

3.3. Example code

`org.hibernate.tutorial.annotations.AnnotationsIllustrationTest` is essentially the same as `org.hibernate.tutorial.hbm.NativeApiIllustrationTest` discussed in Example code.

3.4. Take it further!

Practice Exercises

- [ ] Add an association to the `Event` entity to model a message thread. Use the *Developer Guide* as a guide.
- [ ] Add a callback to receive notifications when an `Event` is created, updated or deleted. Try the same with an event listener. Use the *Developer Guide* as a guide.

4. Tutorial Using the Java Persistence API (JPA)

This tutorial is located within the download bundle under `entitymanager/` .

Objectives

- ✓ Bootstrap a JPA `EntityManagerFactory`
- ✓ Use annotations to provide mapping information
- ✓ Use JPA API calls

4.1. persistence.xml

The previous tutorials used the Hibernate-specific `hibernate.cfg.xml` configuration file. JPA, however, defines a different bootstrap process that uses its own configuration file named `persistence.xml` . This bootstrapping process is defined by the JPA specification. In Java™ SE environments the persistence provider (Hibernate in this case) is required to locate all JPA configuration files by classpath lookup of the `META-INF/persistence.xml` resource name.

Example 10. persistence.xml

```xml
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
        version="2.0">
    <persistence-unit name="org.hibernate.tutorial.jpa">
        ...
    </persistence-unit>
</persistence>
```

`persistence.xml` files should provide a unique name for each "persistence unit". Applications use this name to reference the configuration when obtaining an `javax.persistence.EntityManagerFactory reference.

The settings defined in the `<properties/>` element are discussed in The Hibernate configuration file. Here the `javax.persistence` -prefixed varieties are used when possible. Notice that the remaining Hibernate-specific configuration setting names are now prefixed with `hibernate.` .

Additionally, the `<class/>` element functions the same as we saw in The Hibernate configuration file.

4.2. The annotated entity Java class

The entity is exactly the same as in The annotated entity Java class.

4.3. Example code

The previous tutorials used the Hibernate native APIs. This tutorial uses the JPA APIs.

Example 11. Obtaining the javax.persistence.EntityManagerFactory

```java
protected void setUp() throws Exception {
        sessionFactory = Persistence.createEntityManagerFactory(
"org.hibernate.tutorial.jpa" );
}
```

Notice again that the persistence unit name is `org.hibernate.tutorial.jpa`, which matches persistence.xml.

Example 12. Saving (persisting) entities

```java
EntityManager entityManager = sessionFactory.createEntityManager();
entityManager.getTransaction().begin();
entityManager.persist( new Event( "Our very first event!", new Date() ) );
entityManager.persist( new Event( "A follow up event", new Date() ) );
entityManager.getTransaction().commit();
entityManager.close();
```

The code is similar to Saving entities. The `javax.persistence.EntityManager` interface is used instead of the `org.hibernate.Session` interface. JPA calls this operation "persist" instead of "save".

Example 13. Obtaining a list of entities

```java
entityManager = sessionFactory.createEntityManager();
entityManager.getTransaction().begin();
List<Event> result = entityManager.createQuery( "from Event", Event.class
).getResultList();
for ( Event event : result ) {
        System.out.println( "Event (" + event.getDate() + ") : " + event.getTitle()
);
}
entityManager.getTransaction().commit();
entityManager.close();
```

Again, the code is pretty similar to what we saw in Obtaining a list of entities.

4.4. Take it further!

Practice Exercises

- ☐ Develop an EJB Session bean to investigate implications of using a container-managed persistence context. Try both stateless and stateful use-cases.
- ☐ Use listeners with CDI-based injection to develop a JMS-based event message hub

5. Tutorial Using Envers

This tutorial is located within the download bundle under `envers/`.

Objectives

- ✓ Annotate an entity as historical
- ✓ Configure Envers
- ✓ Use the Envers APIs to view and analyze historical data

5.1. persistence.xml

This file was discussed in the JPA tutorial in persistence.xml, and is essentially the same here.

5.2. The annotated entity Java class

Again, the entity is largely the same as in The annotated entity Java class. The major difference is the addition of the `@org.hibernate.envers.Audited` annotation, which tells Envers to automatically track changes to this entity.

5.3. Example code

The code saves some entities, makes a change to one of the entities and then uses the Envers API to pull back the initial revision as well as the updated revision. A revision refers to a historical snapshot of an entity.

Example 14. Using the `org.hibernate.envers.AuditReader`

```java
public void testBasicUsage() {
    ...
    AuditReader reader = AuditReaderFactory.get( entityManager );
    Event firstRevision = reader.find( Event.class, 2L, 1 );
    ...
    Event secondRevision = reader.find( Event.class, 2L, 2 );
    ...
}
```

We see that an `org.hibernate.envers.AuditReader` is obtained from the `org.hibernate.envers.AuditReaderFactory` which wraps the `javax.persistence.EntityManager`.

Next, the `find` method retrieves specific revisions of the entity. The first call says to find revision number 1 of Event with id 2. The second call says to find revision number 2 of Event with id 2.

5.4. Take it further!

Practice Exercises

- ☑ Provide a custom revision entity to additionally capture who made the changes.
- ☑ Write a query to retrieve only historical data which meets some criteria. Use the *User Guide* to see how Envers queries are constructed.
- ☑ Experiment with auditing entities which have various forms of relationships (many-to-one, many-to-many, etc). Try retrieving historical versions (revisions) of such entities and navigating the object tree.