Hibernate Tips

More than 70 solutions to common Hibernate problems

Thorben Janssen

Hibernate Tips

More than 70 solutions to common Hibernate problems

Thorben Janssen

Hibernate Tips: More than 70 solutions to common Hibernate problems

© 2017 Thorben Janssen. All rights reserved.
Thorben Janssen
Hohe Str. 34
01187 Dresden
Germany
http://www.thoughts-on-java.org

ISBN: 978-1544869179

Copy Editor: Nermina Miller
Cover: ebokks, Hildesheim
Cover Image: BillionPhotos.com – fotolia.com

Table of Contents

Foreword

Undoubtedly, Hibernate ORM and JPA have a steep learning curve. You develop a quick prototype, add a few annotations to your Java classes and everything just works --- things seem easy. But, as you try to tackle more complex mappings or resolve performance problems, you quickly realize that you need a deeper understanding of Hibernate to implement a complete and efficient database access layer.

How do you obtain this deeper understanding? The Hibernate documentation is useful, and we always suggest that users read it all. But, that is a daunting task because the documentation contains a lot of content. Also, the structure of the documentation is to describe individual parts of the solutions. However, object/relational mapping is a very complex concept --- it is simply not feasible for a manual to cover all combinations that are often needed to solve real-world problems and implement real-world use cases.

For example, you might not remember the combination of annotations needed to define a specific mapping idea, or you're just wondering how to implement a specific use case. You need a recipe or a quick tip on how to implement the task you're currently working on. For such cases, users have many options to find solutions including Hibernate's blogs, its user forums, its IRC channels, its HipChat rooms, StackOverflow, and so on. Another great resource are the numerous books on using Hibernate. Additionally a number of blogs exist, dedicated to using Hibernate by community experts --- long-time Hibernate power users. Many of these community expert blogs focus on showing how to use Hibernate's existing features and annotations to implement specific use cases or how to research solving performance problems.

Thorben has been part of this community expert group for a long time, helping Hibernate users via his blog posts, articles, and various forums. And now he has written a book. And, as always, Thorben has a lot of great Hibernate insight to share. This is the first book on Hibernate I have seen that takes an FAQ-style approach, which is an unusual structure. Other books on Hibernate, as well as the Hibernate documentation itself, take the same basic approach to teaching --- they explain the individual pieces in detail, sequentially. While this is valuable (and I'd argue critical) knowledge, it is often hard for new users to apply this sequential, segmented knowledge to resolve more complex topics.

The FAQ approach makes it easier for users to find help on common higher-level concepts and topics. Both forms of knowledge are useful in learning Hibernate. Together with the other listed resources, this book will be a great addition to every developer's Hibernate toolbox.

Steve Ebersole
Lead Developer - Hibernate ORM
Principal Software Engineer - Red Hat, Inc.

Preface

Hibernate is one of the most popular Java Persistence API (JPA) implementations and also one of the most popular Java Object Relational Mapping (ORM) frameworks in general. It helps you to map the classes of your domain model to database tables and automatically generate SQL statements to update the database on object state transitions. That is a complex task, but Hibernate makes it look easy. You just annotate your domain classes, and Hibernate takes care of the rest. Or, it at least seems like that in the beginning.

When you've used Hibernate for a while, you begin to recognize that you need to do more than just add an `@Entity` annotation to your domain model classes. Real-world applications often require advanced mappings, complex queries, custom data types, and caching.

Hibernate can do all of that. You just have to know which annotations and APIs to use. The acute need for this knowledge prompted me to write the Hibernate Tips series on my Thoughts on Java [http://www.thoughts-on-java.org] blog in 2016. In this book, you'll find more than 35 exclusive tips and the most popular tips from the blog.

What you get in this book

More than 70 Hibernate tips show you how to solve different problems with Hibernate. Each of these tips consists of one or more code samples and an easy-to-follow procedure. You can also download [http://www.hibernate-tips.com/download-examples] an example project with executable test cases for each Hibernate tip. I recommend downloading this project before you start reading the book so that you can try each Hibernate tip when you read it.

To help you find the tip for your development task, I grouped them into the following chapters:

- I show you how to bootstrap Hibernate in different execution environments in the Setting up Hibernate chapter.
- In the Basic Mappings chapter, I introduce you to basic attribute mappings that allow you to use Hibernate's standard mappings to map an entity to a database table.

- The tips in the Advanced Mappings chapter show you some of Hibernate's advanced features and how you can use them for things like defining custom mappings for unsupported data types, mapping of read-only database views, defining derived primary keys, and mapping of inheritance hierarchies.
- Hibernate implements the JPA specification, but it also provides several proprietary features. I show you some of them in the Hibernate Specific Queries and Mappings chapter.
- Java 8 introduced several new APIs and programming concepts. Since version 5, you can use them with Hibernate. I show you a few examples in the Java 8 chapter.
- Logging is an important topic that gets ignored in a lot of projects. You should always make sure that Hibernate logs useful information during development and doesn't slow down your application in production. I give you several configuration tips in the Logging chapter.
- The tips in the JPQL chapter show you how to use JPA's query language to read records from the database and how you can use it to update or delete multiple entities at once.
- If your queries are too complex for JPQL, take a look at the Native SQL Queries chapter, which shows how to perform native SQL queries with Hibernate.
- The Criteria API provides another option to create database queries. It is especially useful if you need to create queries programmatically. I show you several examples using this API in the Create queries programmatically with the Criteria API chapter.
- In the Stored Procedures chapter, I explain how you can use the `@NamedStoredProcedureQuery` annotation and the `StoredProcedureQuery` interface to execute stored procedures in your database.
- Caching can be an effective approach to improve the performance of your application. I show you how to activate and use Hibernate's second-level and query cache in the Caching chapter.

How to get the example project

I use a lot of code samples in this book to show you how to solve a specific problem with Hibernate. You can download an example project with all code samples and executable test cases at http://www.hibernate-tips.com/download-examples.

Who this book is for

This book is for developers who are already working with Hibernate and who are looking for solutions for their current development tasks or problems. The tips are designed as self-contained recipes that provide specific solutions and can be accessed as needed. Most tips contain links to related tips that you can follow if you want to dive deeper into a topic or need a slightly different solution. There is no need to read the tips in a specific order. Feel free to read the book from cover to cover or just pick the tips that help you in your current project.

To get the most out of this book, you should already be familiar with the general concepts of JPA and Hibernate. You're in the right place if you are looking for tips on how to use Hibernate to implement your business requirements. I don't explain Hibernate's general concepts, and therefore this book is not intended for beginners. But, if you're already familiar with ORM frameworks and like to learn by doing, you may find this example-based approach helpful.

Setting up Hibernate

You can use Hibernate in several different environments. You can use it as a JPA implementation in a Java SE or Java EE environment, as a proprietary persistence framework in Java SE or as a persistence provider in Spring. The core Hibernate features that I explain in the Hibernate Tips in this book, are available in all these environments. The only differences are features that other frameworks in your environment provide on top of Hibernate and the bootstrapping mechanism.

You can find examples for the different bootstrapping approaches in the following Hibernate tips:

- How to bootstrap Hibernate in a Java SE environment
- How to bootstrap Hibernate in a Java EE environment
- How to use Hibernate's native bootstrapping API
- How to bootstrap Hibernate with Spring Boot
- How to access Hibernate APIs from JPA
- How to automatically add Metamodel classes to your project

How to bootstrap Hibernate in a Java SE environment

Problem

I want to use Hibernate as my JPA provider in a Java SE environment. How do I bootstrap Hibernate?

Solution

Before you can bootstrap Hibernate in your Java SE application, you need to add the required dependencies to your classpath. I'm using Hibernate 5.2.8.Final for the examples of this book, and the `hibernate-core.jar` file is the only required Hibernate dependency. The JPA jar-file is included as a transitive dependency of `hibernate-core`.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.8.Final</version>
</dependency>
```

You also need to add a database-specific JDBC driver to the classpath of your application. Please check your database documentation for more information.

After you've added the required dependencies, you can bootstrap Hibernate as it is defined in the JPA specification. You need to add a `persistence.xml` file to the `META-INF` directory of your application. The following code snippet shows a simple example of a `persistence.xml` file. It configures a persistence-unit with the name `my-persistence-unit`. It also tells Hibernate to use the `PostgreSQLDialect` and to connect to a PostgreSQL database on localhost. Your configuration might differ if you use a different database or a connection pool.

```
<persistence>
    <persistence-unit name="my-persistence-unit">
        <description>Hibernate Tips</description>
        <provider>
            org.hibernate.jpa.HibernatePersistenceProvider
        </provider>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>

        <properties>
            <property name="hibernate.dialect"
                value="org.hibernate.dialect.PostgreSQLDialect" />

            <property name="javax.persistence.jdbc.driver"
                value="org.postgresql.Driver" />
            <property name="javax.persistence.jdbc.url"
                value="jdbc:postgresql://localhost:5432/recipes" />
            <property name="javax.persistence.jdbc.user"
                value="postgres" />
            <property name="javax.persistence.jdbc.password"
                value="postgres" />
        </properties>
    </persistence-unit>
</persistence>
```

You can then call the `createEntityManagerFactory` of the `Persistence` class to create an `EntityManagerFactory` for the `persistence-unit` you configured in your `persistence.xml` file. The `EntityManagerFactory` provides a method to get an `EntityManager`, which I use in most examples in this book. That’s all you need to do to bootstrap Hibernate in your application.

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("my-persistence-unit");
EntityManager em = emf.createEntityManager();
```

Source Code

You can find a project with executable test cases for this Hibernate Tip in the `JPABootstrapping` module of the example project. If you haven't already done so, you can download it at http://www.hibernate-tips.com/download-examples.

Learn More

JPA also defines a bootstrapping approach for Java EE environments. I explain it in How to bootstrap Hibernate in a Java EE environment.

You can also use Hibernate's proprietary bootstrapping API, which gives you access to proprietary configuration features. I show you how to do that in How to use Hibernate's native bootstrapping API.

If you want to use Hibernate with Spring Boot, take a look at How to bootstrap Hibernate with Spring Boot.

How to bootstrap Hibernate in a Java EE environment

Problem

I want to use Hibernate as my JPA provider in a Java EE environment. How do I bootstrap Hibernate?

Solution

Bootstrapping Hibernate in a Java EE environment is pretty simple. You need to make sure that Hibernate is set up in your Java EE application server. That's the case if you're using a JBoss Wildfly or JBoss EAP server. Please check your application server documentation if you're using a different server. Your server might already use Hibernate as the JPA implementation or you need to decide if you want to replace the existing JPA implementation.

You can then bootstrap Hibernate as it is defined in the JPA specification. You just need to add a `persistence.xml` file to the `META-INF` directory of a deployment unit. The following code snippet shows a simple example of a `persistence.xml` file that defines the persistence-unit `my-persistence-unit`. It tells Hibernate to use the `PostgreSQLDialect` and to connect to a PostgreSQL database on localhost. Your configuration might differ if you use a connection pool provided by your application server.

```xml
<persistence>
    <persistence-unit name="my-persistence-unit">
        <description>Hibernate Tips</description>
        <provider>
            org.hibernate.jpa.HibernatePersistenceProvider
        </provider>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>

        <properties>
            <property name="hibernate.dialect"
                value="org.hibernate.dialect.PostgreSQLDialect" />

            <property name="javax.persistence.jdbc.driver"
                value="org.postgresql.Driver" />
            <property name="javax.persistence.jdbc.url"
                value="jdbc:postgresql://localhost:5432/recipes" />
            <property name="javax.persistence.jdbc.user"
                value="postgres" />
            <property name="javax.persistence.jdbc.password"
                value="postgres" />
        </properties>
    </persistence-unit>
</persistence>
```

The container creates an `EntityManagerFactory` for each `persistence-unit` defined in the configuration. It also enables you to inject an `EntityManagerFactory` or an `EntityManager` when you need it.

```java
@PersistenceUnit
private EntityManagerFactory emf;

@PersistenceUnit
private EntityManager em;
```

Learn More

JPA and Hibernate also provide two approaches to bootstrap Hibernate in a Java SE environment. I explain them in How to bootstrap Hibernate in a Java SE environment and How to use Hibernate's native bootstrapping API.

If you want to use Hibernate with Spring Boot, take a look at How to bootstrap Hibernate with Spring Boot.

How to use Hibernate's native bootstrapping API

Problem

I need more control over Hibernate's internal configuration. How do I use its native bootstrapping API?

Solution

Hibernate's native bootstrapping API is very flexible, which makes it more complicated to use but also more powerful than the JPA bootstrapping API. If you don't need this flexibility, I recommend using the JPA API.

Before you can start the bootstrapping process, you need to add the required dependencies to your classpath. I'm using Hibernate 5.2.8.Final for the examples of this book, and the `hibernate-core.jar` file is the only required Hibernate dependency. It also includes the JPA jar-file as a transitive dependency.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.8.Final</version>
</dependency>
```

You also need to add a database-specific JDBC driver to the classpath of your application. Please check your database documentation for more information.

As soon as you add the required dependencies, you can implement the bootstrapping process. You need to create a `StandardServiceRegistry`, build a `Metadata` object, and use that object to instantiate a `SessionFactory`.

Hibernate uses two service registries --- the `BootstrapServiceRegistry` and the `StandardServiceRegistry`. The default `BootstrapServiceRegistry` provides a good solution for most applications, so I skip the programmatic definition in this example.

However, you need to configure the StandardServiceRegistry. In this example, I do that using a hibernate.cfg.xml file, which makes the implementation easy and allows you to change the configuration without changing the source code. Hibernate loads the configuration file automatically from the classpath when you call the configure method on the StandardServiceRegistryBuilder. You can then adapt the configuration programmatically before you call the build method to get a ServiceRegistry.

```
ServiceRegistry standardRegistry =
        new StandardServiceRegistryBuilder()
                .configure()
                .build();
```

The following code snippet shows an example of a hibernate.cfg.xml configuration file. It tells Hibernate to use the PostgreSQLDialect and to connect to a PostgreSQL database on localhost. It also tells Hibernate to generate the database tables based on the entity mappings. Your configuration may differ if you use a different database or a connection pool.

⚠ Generating your database tables based on entity mappings is not recommended for production. You should use SQL scripts instead so that you are in control of your database model and can optimize it for your requirements.

```
<hibernate-configuration>
    <session-factory>
        <property name="dialect">
            org.hibernate.dialect.PostgreSQLDialect
        </property>

        <property name="connection.driver_class">
            org.postgresql.Driver
        </property>
        <property name="connection.url">
            jdbc:postgresql://localhost:5432/recipes
        </property>
        <property name="connection.username">postgres</property>
        <property name="connection.password">postgres</property>
        <property name="connection.pool_size">1</property>

        <property name="hbm2ddl.auto">create</property>
    </session-factory>
</hibernate-configuration>
```

After you instantiate a configured `ServiceRegistry`, you need to create a `Metadata` representation of your domain model. You can do that based on the configuration files `hbm.xml` and `orm.xml` or annotated entity classes. I use annotated classes in the following code snippet. I first use the `ServiceRegistry`, which I created in the previous step to instantiate a new `MetadataSources` object. Then I add my annotated entity classes and call the `buildMetadata` to create the `Metadata` representation. In this example, I use only the `Author` entity. After that, I call the `buildSessionFactory` method on the `Metadata` object to instantiate a `SessionFactory`.

```
SessionFactory sessionFactory =
        new MetadataSources(standardRegistry)
                .addAnnotatedClass(Author.class)
                .buildMetadata()
                .buildSessionFactory();
Session session = sessionFactory.openSession();
```

That is all you need to do to create a basic Hibernate setup with its native API. You can now use the `SessionFactory` to open a new `Session` and use it to read or persist entities.

```
Author a = new Author();
a.setFirstName("Thorben");
a.setLastName("Janssen");
session.persist(a);
```

Source Code

You can find a project with executable test cases for this Hibernate tip in the `HibernateBootstrapping` module of the example project. If you haven't already done so, you can download it at http://www.hibernate-tips.com/download-examples.

Learn More

The bootstrapping API defined by the JPA standard is easier to use but not as flexible. I explain it in more detail in:

- How to bootstrap Hibernate in a Java SE environment
- How to bootstrap Hibernate in a Java EE environment

You can also use Hibernate with Spring Boot. I explain the required bootstrapping process in How to bootstrap Hibernate with Spring Boot.

How to bootstrap Hibernate with Spring Boot

Problem

How do I use Hibernate in my Spring Boot application?

Solution

Spring Boot makes it extremely easy to bootstrap Hibernate. You just need to add the Spring Boot JPA starter to your classpath, and Spring Boot handles the bootstrapping for you.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

You also need to add a database-specific JDBC driver to the classpath of your application. Please check your database documentation for more information.

You define your data source with a few properties in the `application.properties` file. The following configuration example defines a data source that connects to a PostgreSQL database on localhost.

```
spring.datasource.url = jdbc:postgresql://localhost:5432/recipes
spring.datasource.username = postgres
spring.datasource.password = postgres
```

If you add an H2, HSQL, or Derby database on the classpath, you can safely omit the configuration, and Spring Boot starts and connects to an in-memory database. You can also add multiple JDBC drivers and an in-memory database to your classpath and use different configurations for different target environments.

That’s all you need to do bootstrap Hibernate in a Spring Boot application. You can now use the @Autowired annotation to inject an EntityManager.

```
@Autowired
private EntityManager em;
```

Source Code

You can find a project with executable test cases for this Hibernate tip in the SpringBootBootstrapping module of the example project. If you haven’t already done so, you can download it at http://www.hibernate-tips.com/download-examples.

Learn More

JPA defines a bootstrapping approach for Java SE and Java EE environments. I explain it in:

- How to bootstrap Hibernate in a Java SE environment
- How to bootstrap Hibernate in a Java EE environment

You can also use Hibernate’s proprietary bootstrapping API, which gives you access to proprietary configuration features. I show you how to do that in How to use Hibernate’s native bootstrapping API.

How to access Hibernate APIs from JPA

Problem

I'm using Hibernate via the `EntityManager` API. Is there a way to access the proprietary Hibernate `Session` and `SessionFactory`?

Solution

Since version 2.0, JPA provides easy access to the APIs of the underlying implementations. `EntityManager` and `EntityManagerFactory` provide an `unwrap` method, which returns the corresponding classes of the JPA implementation. In Hibernate's case, `Session` and `SessionFactory` give you full access to proprietary Hibernate features, such as the support for `Streams` and `Optional`.

The following code snippet shows you how to get the Hibernate `Session` from `EntityManager`. You just need to call the `unwrap` method on `EntityManager` and provide the `Session` class as a parameter.

```
Session session = em.unwrap(Session.class);
```

As you can see in the next code snippet, you can get Hibernate's `SessionFactory` in a similar way. You first get `EntityMangerFactory` from `EntityManager` and then call the unwrap method with the `SessionFactory` class.

```
SessionFactory sessionFactory = em.getEntityManagerFactory()
                                    .unwrap(SessionFactory.class);
```

Source Code

You can find a project with executable test cases for this Hibernate tip in the `AccessHibernateApi` module of the example project. If you haven't already done so, you can download it at http://www.hibernate-tips.com/download-examples.

Learn More

You also get direct access to Hibernate's `Session` and `SessionFactory` classes, if you use its native bootstrapping API. For more information, see How to use Hibernate's native bootstrapping API.

How to automatically add Metamodel classes to your project

Problem

I use Hibernate's Static Metamodel Generator to generate the JPA Metamodel. These classes are generated to a different directory, which isn't used as a source folder. Is there a way to automatically register this folder as a source folder?

Solution

During my research, I learned from Frits Walraven that there is a Maven plugin that can do exactly that. Special thanks to Frits, who also reviewed this book.

The only thing you need to do is to add the following Maven plugin to your build configuration. It registers a list of directories as additional source folders. I use it in the parent `pom.xml` file of my project to add the directory, to which the JPA Metamodel classes get generated (`target/generated-sources/annotations`), as a source folder.

```
<project>
    ...

    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>build-helper-maven-plugin</artifactId>
                <version>3.0.0</version>
                <executions>
                    <execution>
                        <id>add-source</id>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>add-source</goal>
                        </goals>
                        <configuration>
                            <sources>
                                <source>
                            target/generated-sources/annotations
                                </source>
                            </sources>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

    ...
</project>
```

Source Code

You can find an example of a complete maven build configuration in the example project. If you haven't already done so, you can download it at http://www.hibernate-tips.com/download-examples.

Learn More

The JPA Metamodel provides a type-safe way to reference entity attributes when you create a `CriteriaQuery` or an `EntityGraph`. I explain it in more detail in How to reference entity attributes in a type-safe way.