Hibernate Tip: Hibernate's native bootstrapping API

Question:

I need more control over Hibernate's internal configuration. How can 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. I, therefore, recommend using the JPA API, if you don't need this flexibility.

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've added the required dependencies, you can implement the bootstrapping process. You need to create a *StandardServiceRegistry*, build a *Metadata* object and use it to instantiate a *SessionFactory*.

Hibernate uses two service registries, the *BootstrapServiceRegistry* and the *StandardServiceRegistry*. The default *BootstrapServiceRegistry* provides a good solution for most applications and I, therefore, skip the programmatic definition of it in this example.

But you need to configure the *StandardServiceRegistry*. I do that in this example with a hibernate.cfg.xml file. It 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 might differ if you use a different database or a connection pool.

WARNING: Generating your database tables based on entity mappings is not recommended for production!

```
<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 instantiated a configured *ServiceRegistry*, you need to create a Metadatarepresentation 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 is done, 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);
```