Bootstrap

The term bootstrapping refers to initializing and starting a software component.

In Hibernate, we are specifically talking about the process of building a fully functional **SessionFactory** instance or **EntityManagerFactory** instance, for JPA. The process is very different for each.

Note: - Bootstrapping refers to the process of building and initializing a ***SessionFactory*** *or* ***EntityManagerFactory***. *Bootstrapping APIs as redesigned in 5.0*

Now during the bootstrap process, you can customize Hibernate behavior.

What is bootstrapping good for?

Answer: - If you ever felt like you need more control over Hibernate's internal configuration this new feature is something you can leverage to achieve this goal. And this is useful if you have a simple project which only needs to use Hibernate and no other JPA framework: with the help of the bootstrapping API, you can get your project up-and-running without much magic.

What is cost of bootstrapping?

Answer: - The usage of the new native bootstrapping API makes the configuration more complex but it is more powerful than the JPA bootstrapping API.

Why is this better than previously?

Answer: - The new features introduced to enable you access to the API through Java code. This means you don't have to rely on sole XML configuration, you can add some configuration in your codebase which can only change if you deliver a new version of the software.

In some cases, this is very useful because your application doesn't have to rely on the rightness of the properties configured through the configuration files – or you can tweak some internal configuration in Hibernate which you don't want to change through an external file.

How many ways we can bootstrap in Hibernate?

Answer: - There are three ways we can bootstrap Hibernate: -

1. Legacy Bootstrapping.[Deprecated]
2. Native Bootstrapping.
3. JPA Bootstrapping.

Describe Legacy Bootstrapping?

Answer: - The legacy way to bootstrap a **SessionFactory** is via the **org.hibernate.cfg.Configuration** object.

Configuration represents, essentially, a single point for specifying all aspects of building the SessionFactory: everything from settings, to mappings, to strategies, etc. I like to think of Configuration as a big pot to which we add a bunch of stuff (mappings, settings, etc) and from which we eventually get a SessionFactory.

You can obtain the Configuration by instantiating it directly. You then specify mapping metadata (XML mapping documents, annotated classes) that describe your applications object model and its mapping to a SQL database.

```
Configuration cfg = new Configuration ()
        // addResource does a classpath resource lookup
          .addResource( "Item.hbm.xml")
          .addResource( "Bid.hbm.xml")
        // calls addResource using "/org/hibernate/auction/User.hbm.xml"
        .addClass( org.hibernate.auction.User.class )
        // parses Address class for mapping annotations
           .addAnnotatedClass( Address.class )
           // reads package-level (package-info.class) annotations in the named
package      .      addPackage( "org.hibernate.auction" )
        .setProperty( "hibernate.dialect", "org.hibernate.dialect.H2Dialect")
        .setProperty( "hibernate.connection.datasource", "java:comp/env/jdbc/test")
              .      .setProperty( "hibernate.order_updates", "true");
```

There are other ways to specify Configuration information, including:

- Place a file named hibernate.properties in a root directory of the classpath
- Pass an instance of java.util.Properties to Configuration#setProperties
- Via a hibernate.cfg.xml file
- System properties using Java -Dproperty=value

Note: - Configuration is **semi-deprecated** but still available for use, in a limited form that eliminates these drawbacks. "Under the covers", Configuration uses the new bootstrapping code, so the things available there are also available here in terms of auto-discovery.

Describe Native Bootstrapping?

Answer: - The first step in native bootstrapping is the building of a **ServiceRegistry** holding the **services** Hibernate will need during bootstrapping and at run time.

The second step in native bootstrapping is the building of an **org.hibernate.boot.Metadata** object containing the parsed representations of an application domain model and its mapping to a database.

The first thing we obviously need to build a parsed representation is the source information to be parsed (annotated classes, hbm.xml files, orm.xml files). This is the purpose of **org.hibernate.boot.MetadataSources.**

Once we have the sources of mapping information defined, we need to build the **Metadata object**.

The final step in native bootstrapping is to build the **SessionFactory** itself.

Describe JPA Bootstrapping?

Answer: - Bootstrapping Hibernate as a JPA provider can be done in a **JPA-spec compliant manner or using a proprietary bootstrapping approach**. The standardized approach has some limitations in certain environments, but aside from those, it is **highly** recommended that you use JPA-standardized bootstrapping.

In JPA, we are ultimately interested in bootstrapping a **javax.persistence.EntityManagerFactory** instance. The JPA specification defines two primary standardized bootstrap approaches depending on how the application intends to access the **javax.persistence.EntityManager** instances from an **EntityManagerFactory.**

It uses the terms **EE** and **SE** for these two approaches,

What the JPA spec calls **EE** bootstrapping implies the existence of a **container (EE, OSGi, etc)**, who'll manage and inject the persistence context on behalf of the application. You can say like that **compliant container-bootstrapping**.

What it calls **SE** bootstrapping is everything else. You can say like that **application-bootstrapping**

compliant container-bootstrapping

For compliant container-bootstrapping, the container will build an **EntityManagerFactory** for each persistent-unit defined in the **META-INF/persistence.xml configuration file** and make that available to the application for injection via the **javax.persistence.PersistenceUnit** annotation or via JNDI lookup.

```
@PersistenceUnitprivate EntityManagerFactory emf;
```

Or, in case you have multiple Persistence Units (e.g., multiple persistence.xml configuration files), you can inject a specific EntityManagerFactory by Unit name:

```
@PersistenceUnit(    unitName = "CRM")private EntityManagerFactory entityManagerFactory;
```

compliant application-bootstrapping

For compliant application-bootstrapping, rather than the container building the **EntityManagerFactory** for the application, the application builds the **EntityManagerFactory** itself using the **javax.persistence.Persistence** bootstrap class. The application creates an **EntityManagerFactory** by calling the **createEntityManagerFactory** method:

```
// Create an EMF for our CRM persistence-unit.EntityManagerFactory emf = Persistence.createEntityManagerFactory( "CRM");
```

To inject the default Persistence Context, you can use the @PersistenceContext annotation.

```
@PersistenceContextprivate EntityManager em;
```

To inject a specific Persistence Context, you can use the @PersistenceContext annotation, and you can even pass EntityManager-specific properties using the @PersistenceProperty annotation.

```
@PersistenceContext(
unitName = "CRM",
properties = {
    @PersistenceProperty(          name="org.hibernate.flushMode",
   value= "MANUAL"       )    })
private EntityManager entityManager;
```

Externalizing XML mapping files

JPA offers two mapping options:

- annotations
- XML mappings

You can even **mix** annotations and XML mappings so that you can override annotation mappings with XML configurations that can be easily changed without recompiling the project source code. This is possible because if there are **two conflicting mappings, the XML mappings take precedence over its annotation counterpart.**

Summary

Bootstrapping: -

1.Native Bootstrapping

- Building the ServiceRegistry
- Building the Metadata
- Building the SessionFactory

2.JPA Bootstrapping

- JPA-compliant bootstrapping
- Proprietary 2-phase bootstrapping