

Hibernate

Object/Relational Mapping and Transparent
Object Persistence for Java and SQL Databases



Facts about Hibernate

- True transparent persistence
- Query language aligned with SQL
- Does not use byte code enhancement
- Free/open source

What is Hibernate?

- ② ** Hibernate is a powerful, ultra-high performance object/relational persistence and query service for Java. Hibernate lets you develop persistent objects following common Java idiom – including association, inheritance, polymorphism, composition and the Java collections framework. Extremely fine-grained, richly typed object models are possible. The Hibernate Query Language, designed as a "minimal" object-oriented extension to SQL, provides an elegant bridge between the object and relational worlds. Hibernate is now the most popular ORM solution for Java.

Object-relational impedance mismatch

- Object databases are not the answer
- Application Objects cannot be easily persisted to relational databases
- Similar to the putting square peg into round hole

Object/Relational Mapping (ORM)

- Mapping application objects to relational database
- Solution for infamous object-relational impedance mismatch
- Finally application can focus on objects

Transparent Persistence

- Persist application objects without knowing what relational database is the target
- Persist application objects without “flattening” code weaved in and out of business logic

Query Service

- Ability to retrieve sets of data based on criteria
- Aggregate operations like count, sum, min, max, etc.

Why use Hibernate?

- Simple to get up and running
- Transparent Persistence achieved using Reflection
- Isn't intrusive to the build/deploy process
- Persistence objects can follow common java idioms: Association, Inheritance, Polymorphism, Composition, Collections
- In most cases Java objects do not even know they can be persisted

Why use Hibernate

cont

- Java developer can focus on object modeling
- Feels much more natural than Entity Beans or JDBC coding
- Query mechanism closely resembles SQL so learning curve is low

What makes up a Hibernate application?

- Standard domain objects defined in Java as POJO's, nothing more.
- Hibernate mapping file
- Hibernate configuration file
- Hibernate Runtime
- Database

What is missing from a Hibernate application?

- Flattening logic in Java code to conform to relational database design
- Inflation logic to resurrect Java object from persistent store
- Database specific tweaks

Hibernate Classes

- ① SessionFactory - One instance per app. Creates Sessions. Consumer of hibernate configuration file.
- ① Session - Conversation between application and Hibernate
- ① Transaction Factory - Creates transactions to be used by application
- ① Transaction - Wraps transaction implementation(JTA, JDBC)

Hibernate Sample App

- Standard Struts Web Application
- Deployed on JBoss
- Persisted to MySQL database
- All hand coded, didn't use automated tools to generate Java classes, DDL, or mapping files
- Disclaimer: Made simple stupid on purpose

DVD Library

- 👁️ Functions

- 👁️ Add DVD's to your collection

- 👁️ Output your collection

Separation of Responsibility

JSP

Struts
Action

Persistence
classes

DVD POJO

```
public class DVD {  
    private Long id;  
    private String name;  
    private String url;  
  
    public Long getId() {  
        return id;  
    }  
  
    public void setId(Long id) {  
        this.id = id;  
    }  
    ..  
    ..  
}
```

NOTE:

I transform
the DVDForm Bean to DVD
prior to persisting for
added flexibility

Hibernate Mapping File

DVD.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>
  <class name="com.shoesobjects.DVD" table="dvdlist">
    <id name="id" column="id" type="java.lang.Long" unsaved-value="null">
      <generator class="native"/>
    </id>
    <property name="name" column="name" type="java.lang.String" not-null="true"/>
    <property name="url" column="url" type="java.lang.String"/>
  </class>
</hibernate-mapping>
```

Hibernate.properties

```
#####  
### Platforms #####  
#####
```

```
## JNDI Datasource  
hibernate.connection.datasource java:/DVDDDB
```

```
## MySQL  
hibernate.dialect net.sf.hibernate.dialect.MySQLDialect  
hibernate.connection.driver_class org.gjt.mm.mysql.Driver  
hibernate.connection.driver_class com.mysql.jdbc.Driver
```

DVDService Class

```
public void updateDVD(DVD dvd) {
    Session session = ConnectionFactory.getInstance().getSession();

    try {
        Transaction tx = session.beginTransaction();
        session.update(dvd);
        tx.commit();
        session.flush();
    } catch (HibernateException e) {
        tx.rollback();
    } finally {
        // Cleanup
    }
}
```

ConnectionFactory

```
public class ConnectionFactory {  
  
    private static ConnectionFactory instance = null;  
    private SessionFactory sessionFactory = null;  
  
    private ConnectionFactory() {  
        try {  
            Configuration cfg = new Configuration().addClass(DVD.class);  
            sessionFactory = cfg.buildSessionFactory();  
        } catch (Exception e) {  
            // Do something useful  
        }  
    }  
  
    public Session getSession() {  
        Session session = null;  
        try {  
            session = sessionFactory.openSession();  
        } catch (HibernateException e) {  
            // Do Something useful  
        }  
        return session;  
    }  
}
```

ConnectionFactory Improved

```
public class ConnectionFactory {  
  
    private static ConnectionFactory instance = null;  
    private SessionFactory sessionFactory = null;  
  
    private ConnectionFactory() {  
        try {  
            Configuration cfg = new Configuration().configure().buildSessionFactory();  
        } catch (Exception e) {  
            // Do something useful  
        }  
    }  
  
    public Session getSession() {  
        Session session = null;  
        try {  
            session = sessionFactory.openSession();  
        } catch (HibernateException e) {  
            // Do Something useful  
        }  
        return session;  
    }  
}
```

Hibernate.cfg.xml

- Alternative to hibernate.properties
- Handles bigger applications better
- Bind SessionFactory to JNDI Naming
- Allows you to remove code like the following and put it in a configuration file
 - `Configuration cfg = new Configuration().addClass(DVD.class);`

Sample hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">
<hibernate-configuration>
  <!-- a SessionFactory instance listed as /jndi/name -->
  <session-factory name="java:comp/env/hibernate/SessionFactory">
    <property name="connection.datasource">java:/SomeDB</property>
    <property name="show_sql">true</property>
    <property name="dialect">net.sf.hibernate.dialect.MySQLDialect</property>
    <property name="use_outer_join">true</property>
    <property name="transaction.factory_class">net.sf.hibernate.transaction.JTATransactionFactory</>
    <property name="jta.UserTransaction">java:comp/UserTransaction</property>

    <!-- Mapping files -->
    <mapping resource="com/shoesobjects/SomePOJO.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Schema Generation

- SchemaExport can generate or execute DDL to generate the desired database schema
- Can also update schema
- Can be called via ant task

Code Generation

- hbm2java
- Parses hibernate mapping files and generates POJO java classes on the fly.
- Can be called via ant task

Mapping File Generation

- MapGenerator – part of Hibernate extensions
- Generates mapping file based on compiled classes. Some rules apply.
- Does repetitive grunt work

Links to live by

- 👁 <http://www.hibernate.org>
- 👁 <http://www.springframework.org>
- 👁 <http://www.jboss.org>
- 👁 [http://raibledesigns.com/wiki/Wiki.jsp?
page=AppFuse](http://raibledesigns.com/wiki/Wiki.jsp?page=AppFuse)