



pointing the way

White Papers

Hibernate

Persistence Framework for
JEE Applications



Introduction

This paper describes a Hibernate framework for accessing persistence data in Java business applications running on JBoss Application Server.

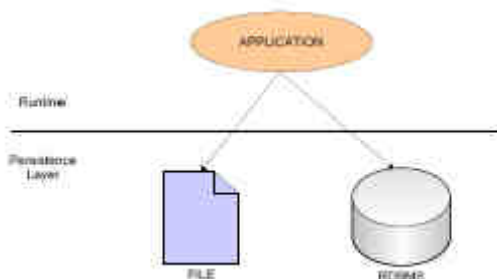
Background

In general, persistence is a data that live longer than application that created it. Data have to be saved between two program executions. The most popular way uses files (XML, serialization) or relational databases.

Business systems mostly work with relational database because RDBMS has numbers of benefits such as: enhanced data integrity, application-data independence, improved security, logical and physical data independent, and many others.

Accessing database from business application can be made by using common frameworks and APIs like: JDBC, JDO, JPA, EJB, ORM. The most popular implementation that combines the best ideas common for JDO and JPA is a Hibernate. It provides support for many persistence mechanisms such as:

- ✓ Java Object and Advanced Object Oriented Concepts,
- ✓ Transactional Integrity and Concurrency,
- ✓ Large Data Processing,
- ✓ High Performance, Dual-Layer Cache Architecture,
- ✓ Queries Language (HQL),
- ✓ J2EE Integration (JCA, JMX support).





Extremely easy of use Hibernate API makes it fast for developers to implement with fewer mistakes in code.

Solution

Basic Terms

To understand Hibernate let us introduce key terms and features:

POJO

Plain Old Java Object is a simple Java Class without any special features like extending prespecified classes or implements prespecified interfaces. POJO is restricted only by Java Language Specification.

Annotations

This is an extra meta-information asserted to source code in the form of comment. It can be used with packages, classes, methods, parameters and attributes. Metadata represented by annotations are usually part of other source code activities.

XML Descriptor

The XML format is used to store a metadata for mapping between Object Oriented World and Relational World. Hibernate allows to use XML Descriptor files (*.hbm.xml) as an alternative to annotations.

Entity

It is a representation of an atomic data in Relational World. This term refers also to POJO with included annotations.

Entity Manager

Entity Manager provides an API to manage entities in persistence context. It handles interactions with database, executes queries and performs mapping between OO and Relational database.

Persistence Unit

It consists of all entities that are available for Entity Manager during application runtime.

Persistence Context

This is set of active entity instances that are currently manipulated by application. Persistence context is created based on information stored in Persistence Unit.

HQL

Hibernate Query Language is an adaptation of SQL language to work with object oriented entities in Java code. Using HQL makes application independent from database vendor's specific SQL implementations.

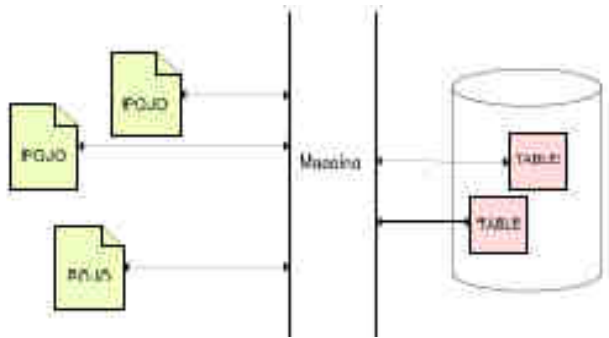
Work Environment

Hibernate is designed to work inside and outside Java Enterprise Edition (Java EE) container. Inside container (e.g. on JBoss Application Server) Hibernate application can use the container to manage persistence, so this makes application even simpler to implement. Outside container Hibernate has to manage persistence by itself. Hibernate is also integrated with known frameworks like Spring, Struts, Grails, Tapestry.

Introducing Entity Manager

Entity Manager is a core part of Hibernate and it is responsible for managing entities including:

- ✓ Mapping between OO and Relational World,
- ✓ Entity Lifecycle (persist, merge, retrieve),
- ✓ Interacting with persistence store (database).



Hibernate makes object-relational mapping by using annotations or XML descriptors to map objects into one or more tables in a database.

Entity Manager provides SQL-like CRUD operations.

Create	When application request to create new entity, the Entity Manager translates entity into a database record.
Read	Entity Manager retrieves rows from database and instantiates a corresponding Java Class entity.
Update	In case of update request, manager tracks down the relational data that match to the entity and updates it.
Delete	When application request to remove entity, manager deletes a corresponding row in database.

Besides of these four basic operations, Entity Manager tries to keep entities synchronized with database as long as they are in persistence context.

Entity Lifecycle

Entity Manager synchronized entities state with database. It means that if application creates new entity or removes one, Manager insert or delete row in given table.



Entity Manager API methods can change actual entity state.

Changes to the entity's data are reflected in database by periodically check for data freshness as long as entity is managed. Detached entities are not managed, so they do not reflect to data in database.

HQL

It is a powerful query language that allows manipulating data represented by entities. Because HQL is derived from SQL its syntax is easy to use and understand for software developers that have basic knowledge of SQL.

General SELECT statement has following syntax:

```
SELECT_clause FROM_clause
[WHERE_clause] [GROUP BY_clause]
[HAVING_clause] [ORDER BY_clause]
```

Update and delete syntax:

```
UPDATE_clause [WHERE_clause]
DELETE_clause [WHERE_clause]
```

Example query finds all players who participate in given sport:

```
SELECT DISTINCT p
FROM Player p, IN (p.team) t
WHERE t.league.sport = :sportParam
```

How It All Works Together

The simplest way to define entity is by using annotations.

Hibernate



Take a look on very simple example.

Let assume that data model contains a table for storing orders.



Java Class that represents related entity may look like this:

```
@Entity
@Table(name = "ORDER")
```

@Entity says that this POJO will be treating like entity. @Table maps to DB table.

```
@NamedQueries({
    @NamedQuery(name =
"findByCustomer", query = "select o
from Order o where o.customer =
:customerIdParam")
})
```

@NamedQueries annotation by using HQL language allows querying DB to find order by given customerId.

```
public class Order {
    @Id
    @Column(name = "ID")
    private Integer orderId;
```

```
@Column(name = "DATE")
    private Date orderDate;
```

Annotations @Column defines a mapping between attributes and database tables columns.

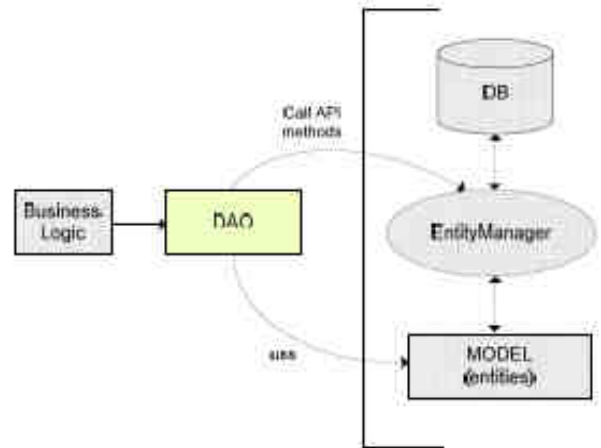
```
@ManyToOne(cascade=CascadeType.MERGE,
fetch=FetchType.EAGER)
@JoinColumn(name="CUSTOMER_ID")
@Column(name = "CUSTOMER_ID") private
Customer customer;
}
```

@ManyToOne defines a relationship between two tables.

Last missing puzzle is a Data Access Object implementation.

Below is an example of save method in DAO class:

```
public void save(Order o) {
    getJpaTemplate().persist(o);
}
```



DAO is a design pattern that provides a isolation between application's business logic and database access implementation. It allows changing data access mechanism or combining data (i.e. due to changes in data model) and keeping the same data access interface for rest of the application.

Hibernate Extra Features

Hibernate Shards

Hibernate Shards is a Hibernate extension for encapsulating and minimizing complexity of working with partitioned data. With Shards its is possible to query and transact against shared datasets using Hibernate Core API.

Hibernate Validator

Hibernate Validator extends an annotations adding constraints for java objects attributes. Validator checks attribute's value for correctness in order to fit it to a type of database table column.



Hibernate Search

This feature extends a Hibernate search mechanism to allow “full text search” by using Apache Lucene project. Hibernate Search indexes domain model using annotations, takes care of database synchronization and brings back managed object from free text queries.

Hibernate Tools

Hibernate Tools provides a Ant tasks and Eclipse plugins for performing reverse engineering, code generation and visualization during typical work with Hibernate framework.

Conclusion

Hibernate framework has a very rich API and it easy to use for relational data. It was designed to provide a simple alternative for persistence in Java

EE platform. It is mostly common used persistence framework that results in combination of the best solutions accepted by developers and proven by test of time. Hibernate is a more advanced technology than Java native persistence mechanism like JDBC or serialization. Hibernate is chosen by IT experts all over the world to build a new generation business applications.

Links

Norman Richards, Sam Griffith Jr. – “JBoss – A Developer’s Notebook”, O’Reilly 2005
Tom Marrs, Scott Davis – “JBoss at Work”, O’Reilly 2006

Hibernate Documentation
<http://www.hibernate.org/5.html>

Using Hibernate with JBoss
<http://www.hibernate.org/66.html>