Research Centers
Core Java
Enterprise Java
Mobile Java
Tools & Methods
JavaWorld Archives

Site Resources
Featured Articles
News & Views
Community
Java Q&A
JW Blogs
Podcasts
Site Map
Careers
Newsletters
Whitepapers
RSS Feeds

About JavaWorld
Advertise
Write for JW

iBATIS, Hibernate, and JPA: Which is right for you?

Object-relational mapping solutions compared

By K. L. Nitin, Ananya S., Mahalakshmi K., and S. Sangeetha, JavaWorld.com, 07/15/08

Print Feedback 24 Comments

Working with JPA

JPA uses many interfaces and annotation types defined in the `javax.persistence` package available with version 5 of Java EE. JPA uses entity classes that are mapped to tables in the database. These entity classes are defined using JPA annotations. Listing 9 shows the entity class named `Employee` that corresponds to the EMPLOYEE table in the sample application's database.

Listing 9. Employee entity class

```
@Entity
@Table(name = "employee")
@NamedQueries({@NamedQuery(name = "Employee.findByEmpId", qu
public class Employee implements Serializable {
        @Id
      @Column(name = "emp_id", nullable = false)
  private Integer empId;
  @Column(name = "emp_firstname", nullable = false)
  private String empFirstname;
  @Column(name = "emp_lastname", nullable = false)
  private String empLastname;

public Employee() {    }
public Employee(Integer empId) {
        this.empId = empId;
}
public Employee(Integer empId, String empFirstname, String e
   this.empId = empId;
        this.empFirstname = empFirstname;
        this.empLastname = empLastname;
}
public Integer getEmpId() {
        return empId;
      }
      public void setEmpId(Integer empId) {
        this.empId = empId;
      }
      public String getEmpFirstname() {
        return empFirstname;
}
      public void setEmpFirstname(String empFirstname) {
        this.empFirstname = empFirstname;
}
      public String getEmpLastname() {
        return empLastname;
```

JW's Most Read
Scala on their minds -- bloggers debate Scala's complexity
Simplify the data access layer with Spring 3.1 and Java generics
Bitcoin for beginners, Part 2: Bitcoin as a technology and network
Why Yammer dropped Scala for Java
Engine Yard vs Heroku: Ruby clouds (for Java too)

Featured White Papers
Observation Driven Testing: What else is your code doing?
Software Agitation: Your Own Personal Code Reviewer
Early & Often: Avoiding Security Flaws with CI, High Coverage

```
}
   /****
*override equals, hashcode and toString methods
*using @Override annotation
******/
```

The features of an entity class are as follows:

- The entity class is annotated using the `javax.persistence.Entity` annotation (`@Entity`).
- It must have a public or protected no-argument constructor, and may also contain other constructors.
- It cannot be declared `final`.
- Entity classes can extend from other entities and non-entity classes as well; the converse is also possible.
- They cannot have `public` instance variables. The class members should be exposed using only `public` getter and setter methods, following JavaBean style.
- Entity classes, being POJOs, generally need not implement any special interfaces. However, if they are to be passed as arguments over the network, then they must implement the `Serializable` interface.

The `javax.persistence.Table` annotation specifies the name of the table to which this entity instance is mapped. The class members can be Java primitive types, wrappers of Java primitives, enumerated types, or even other embeddable classes. The mapping to each column of the table is specified using the `javax.persistence.Column` annotation. This mapping can be used with persistent fields, in which case the entity uses persistent fields as well, or with getter/setter methods, in which case the entity uses persistent properties. However, the same convention must be followed for a particular entity class. Also, fields that are annotated using the `javax.persistence.Transient` annotation or marked `transient` will not be persisted into the database.

Each entity has a unique object identifier. This identifier is used to differentiate among different entity instances in the application domain; it corresponds to a primary key that is defined in the corresponding table. A primary key can be simple or composite. A simple primary key is denoted using the `javax.persistence.Id` annotation. Composite primary keys can be a single persistent property/field or a set of such fields/properties; they must be defined in a primary key class. Composite primary keys are denoted using the `javax.persistence.EmbeddedId` and `javax.persistence.IdClass` annotations. Any primary key class should implement the `hashcode()` and `equals()` methods.

The life cycle of JPA entities is managed by the entity manager, which is an instance of `javax.persistence.EntityManager`. Each such entity manager is associated with a persistence context. This context can be either propagated across all application components or managed by the application. The `EntityManager` can be created in the application using an `EntityManagerFactory`, as shown in Listing 10.

Listing 10. Creating an EntityManager

```
public class Main {
  private EntityManagerFactory emf;
  private EntityManager em;
  private String PERSISTENCE_UNIT_NAME = "EmployeePU";
  public static void main(String[] args) {
      try      {
          Main main = new Main();
          main.initEntityManager();
          main.create();
          System.out.println("Employee successfully added");
          main.closeEntityManager();
      }
      catch(Exception ex)      {
          System.out.println("Error in adding employee");
          ex.printStackTrace();
      }
  }
 private void initEntityManager()  {
 emf = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);
         em = emf.createEntityManager();
  }
  private void closeEntityManager()   {
      em.close();    emf.close(); }
 private void create() {
      em.getTransaction().begin();
      Employee employee=new Employee(100);
      employee.setEmpFirstname("bob");
      employee.setEmpLastname("smith");
      em.persist(employee);
      em getTransaction() commit();
```

The `PERSISTENCE_UNIT_NAME` represents the name of the persistence unit that is used to create the `EntityManagerFactory`. The `EntityManagerFactory` can also be propagated across the application components using the `javax.persistence.PersistenceUnit` annotation.

In the `create()` method in Listing 10, a new employee record is being inserted into the EMPLOYEE table. The data represented by the entity instance is persisted into the database once the `EntityTransaction` associated with `persist()` is completed. JPA also defines static and dynamic queries to retrieve the data from the database. Static queries are written using the `javax.persistence.NamedQuery` annotation, as shown in the `Employee` entity class. Dynamic queries are defined directly in the application using the `createQuery()` method of the `EntityManager`.

< Prev 1 2 3 4 5 6 7 Next >

Resources

Learn more about the three technologies discussed in this article from the project homepages:

- Hibernate
- iBATIS
- The Java Persistence API

"Get started with Hibernate" (Christian Bauer and Gavin King, JavaWorld, October 2004) is a short introduction to Hibernate written by its creator, Gavin King. *Excerpted from Hibernate in Action; Manning 2004.*)

Java Persistence with Hibernate (Christian Bauer and Gavin King; Manning, November 2006) is the updated second edition of *Hibernate in Action*. Also see iBATIS in Action (Clinton Begin, Brandon Goodin, and Larry Meadors, 2007).

For broader coverage of Java persistence solutions including JDBC, OpenJPA, and pureQuery, see Persistence in the Enterprise: A Guide to Persistence Technologies (Roland Barcia, Geoffrey Hambrick, Kyle Brown, Robert Peterson, Kulvir Singh Bhogal; IBM Press, May 2008).

Ted Neward introduces the so-called object-relational impedance mismatch in his blog post "The Vietnam of computer science."

"Flexible reporting with JasperReports and iBatis" (Scott Monahan, JavaWorld, December 2007) is a hands-on introduction to the iBatis Data Mapper framework.

"Understanding the Java Persistence API" (Aditi Das, JavaWorld, January 2008) is a two-part introduction to Java-platform persistence with OpenJPA.

Java-source.net lists a roundup of open source persistence frameworks for Java.

Visit the JavaWorld Java Enterprise Edition research center for more articles about enterprise data management and Java persistence solutions.

Also see Network World's IT Buyer's Guides: Side-by-side comparison of hundreds of products in over 70 categories.