## JPA介绍：

  JPA全称为[**Java**](http://lib.csdn.net/base/java) Persistence API ，[**Java**](http://lib.csdn.net/base/javase)持久化API是Sun公司在[**Java EE**](http://lib.csdn.net/base/javaee) 5规范中提出的Java持久化接口。JPA吸取了目前Java持久化技术的优点，旨在规范、简化Java对象的持久化工作。使用JPA持久化对象，并不是依赖于某一个ORM框架。

## 为什么要使用JAP?

  在说为什么要使用JPA之前，我们有必要了解为什么要使用ORM技术。

ORM 是Object-Relation-Mapping，即对象关系映射技术，是对象持久化的核心。ORM是对JDBC的封装，从而解决了JDBC的各种存在问题：

a) 繁琐的代码问题

用JDBC的API编程访问[**数据库**](http://lib.csdn.net/base/mysql)，代码量较大，特别是访问字段较多的表的时候，代码显得繁琐、累赘，容易出错。例如：PreparedStatement pstmt=con.prepareStatment("insert into account value(?,?,?,?,?,?,?,?,?)");

ORM则建立了Java对象与数据库对象之间的影射关系，程序员不需要编写复杂的SQL语句，直接操作Java对象即可，从而大大降低了代码量，也使程序员更加专注于业务逻辑的实现。

b) 数据库对象连接问题

关系数据对象之间，存在各种关系，包括1对1、1对多、多对1、多对多、级联等。在数据库对象更新的时候，采用JDBC编程，必须十分小心处理这些关系，以保证维持这些关系不会出现错误，而这个过程是一个很费时费力的过程。

ORM建立Java对象与数据库对象关系影射的同时，也自动根据数据库对象之间的关系创建Java对象的关系，并且提供了维持这些关系完整、有效的机制。

c) 系统[**架构**](http://lib.csdn.net/base/architecture)问题

JDBC属于数据访问层，但是使用JDBC编程时，必须知道后台是用什么数据库、有哪些表、各个表有有哪些字段、各个字段的类型是什么、表与表之间什么关系、创建了什么索引等等与后台数据库相关的详细信息。

使用ORM技术，可以将数据库层完全隐蔽，呈献给程序员的只有Java的对象，程序员只需要根据业务逻辑的需要调用Java对象的Getter和 Setter方法，即可实现对后台数据库的操作，程序员不必知道后台采用什么数据库、有哪些表、有什么字段、表与表之间有什么关系。

d) 性能问题

采用JDBC编程，在很多时候存在效率低下的问题。

pstmt =conn.prepareStatement("insert into user\_info values(?,?)");  
       for (int i=0; i<1000; i++) {  
          pstmt.setInt(1,i);  
          pstmt.setString(2,"User"+i.toString());  
          pstmt.executeUpdate();  
       }

以上程序将向后台数据库发送1000次SQL语句执行请求，运行效率较低。

采用ORM技术，ORM框架将根据具体数据库操作需要，会自动延迟向后台数据库发送SQL请求，ORM也可以根据实际情况，将数据库访问操作合成，尽量减少不必要的数据库操作请求。

## JPA的优势

JPA是目前比较流行的一种ORM技术之一，所以他拥有ORM技术的各种特点，当然他还有自己的一些优势：

### 1 标准化

JPA 是 JCP 组织发布的 Java EE 标准之一，因此任何声称符合 JPA 标准的框架都遵循同样的架构，提供相同的访问 API，这保证了基于JPA开发的企业应用能够经过少量的修改就能够在不同的JPA框架下运行。

### 2 对容器级特性的支持

JPA 框架中支持[**大数据**](http://lib.csdn.net/base/hadoop)集、事务、并发等容器级事务，这使得 JPA 超越了简单持久化框架的局限，在企业应用发挥更大的作用。

### 3 简单易用，集成方便

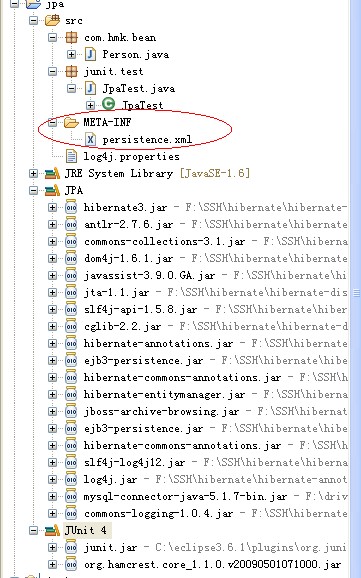
JPA的主要目标之一就是提供更加简单的编程模型：在JPA框架下创建实体和创建Java 类一样简单，没有任何的约束和限制，只需要使用 javax.persistence.Entity进行注释；JPA的框架和接口也都非常简单，没有太多特别的规则和设计模式的要求，开发者可以很容易的掌握。JPA基于非侵入式原则设计，因此可以很容易的和其它框架或者容器集成。

### 4 可媲美JDBC的查询能力

JPA的查询语言是面向对象而非面向数据库的，它以面向对象的自然语法构造查询语句，可以看成是[**hibernate**](http://lib.csdn.net/base/javaee) HQL的等价物。JPA定义了独特的JPQL（Java Persistence Query Language），JPQL是EJB QL的一种扩展，它是针对实体的一种查询语言，操作对象是实体，而不是关系数据库的表，而且能够支持批量更新和修改、JOIN、GROUP BY、HAVING 等通常只有 SQL 才能够提供的高级查询特性，甚至还能够支持子查询。  
5 支持面向对象的高级特性  
　　JPA 中能够支持面向对象的高级特性，如类之间的继承、多态和类之间的复杂关系，这样的支持能够让开发者最大限度的使用面向对象的模型设计企业应用，而不需要自行处理这些特性在关系数据库的持久化。

## 具体例子

  完成工程后工程结构如下图：



**(注意：persistence.xml文件的位置决定持久性的根(Persistence Root)。持久性的根为JAR文件或者包含META-INF目录（前提是persistence.xml位于此）的目录。一般将这个persistence.xml文件放在src下的META-INF中。命名及位置都不能变)**

a)、导入相关jar包（见上图）和创建META-INF和persistence.xml文件.

persistence.xml配置如下：(我用的数据为[**MySQL**](http://lib.csdn.net/base/mysql),采用不同数据库及JPA的不同实现版本会导致配置内容不同)

**[xhtml]** [view plain](http://blog.csdn.net/hmk2011/article/details/6289151) [copy](http://blog.csdn.net/hmk2011/article/details/6289151)

<persistence xmlns="http://java.sun.com/xml/ns/persistence"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_version=1">

    <persistence-unit name="mysqlJPA" transaction-type="RESOURCE\_LOCAL">

        <properties>

            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect" />

            <property name="hibernate.connection.driver\_class" value="com.mysql.jdbc.Driver" />

            <property name="hibernate.connection.username" value="root" />

            <property name="hibernate.connection.password" value="123456" />

            <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/db1" />

            <property name="hibernate.max\_fetch\_depth" value="3" />

            <property name="hibernate.hbm2ddl.auto" value="update" />

        </properties>

    </persistence-unit>

</persistence>

b）、编写实体bean，如下：

**[c-sharp]** [view plain](http://blog.csdn.net/hmk2011/article/details/6289151) [copy](http://blog.csdn.net/hmk2011/article/details/6289151)

1. package com.hmk.bean;
3. import javax.persistence.Column;
4. import javax.persistence.Entity;
5. import javax.persistence.GeneratedValue;
6. import javax.persistence.Id;
8. @Entity
9. public class Person {
10. private int id;
11. private String name;
13. @Id @GeneratedValue
14. public int getId() {
15. return id;
16. }
17. public void setId(int id) {
18. this.id = id;
19. }
20. @Column(length=12)
21. public String getName() {
22. return name;
23. }
24. public void setName(String name) {
25. this.name = name;
26. }
28. }

c）、编写junit[**测试**](http://lib.csdn.net/base/softwaretest)代码，如下：

**[java]** [view plain](http://blog.csdn.net/hmk2011/article/details/6289151) [copy](http://blog.csdn.net/hmk2011/article/details/6289151)

1. package junit.test;

4. import javax.persistence.EntityManager;
5. import javax.persistence.EntityManagerFactory;
6. import javax.persistence.Persistence;
8. import org.junit.BeforeClass;
9. import org.junit.Test;
11. import com.hmk.bean.Person;
13. public class JpaTest {
15. @BeforeClass
16. public static void setUpBeforeClass() throws Exception {
17. }
19. @Test public void createTable(){
20. //可以验证生成表是否正确
21. EntityManagerFactory factory = Persistence.createEntityManagerFactory("mysqlJPA");
22. factory.close();
23. }
25. @Test public void save(){
26. EntityManagerFactory factory = Persistence.createEntityManagerFactory("mysqlJPA");
27. EntityManager em = factory.createEntityManager();
28. em.getTransaction().begin();
29. Person person = new Person(); //person为new状态
30. person.setName("zhang san");
31. em.persist(person); //持久化实体
32. em.getTransaction().commit();
33. em.close();
34. factory.close();
35. }
36. //new 、托管、脱管、删除
38. @Test public void update(){
39. EntityManagerFactory factory = Persistence.createEntityManagerFactory("mysqlJPA");
40. EntityManager em = factory.createEntityManager();
41. em.getTransaction().begin();
42. Person person = em.find(Person.class, 1);
43. person.setName("hmk"); //person为托管状态
44. em.getTransaction().commit();
45. em.close();
46. factory.close();
47. }
49. @Test public void update2(){
50. EntityManagerFactory factory = Persistence.createEntityManagerFactory("mysqlJPA");
51. EntityManager em = factory.createEntityManager();
52. em.getTransaction().begin();
53. Person person = em.find(Person.class, 1);
54. em.clear(); //把实体管理器中的所有实体变为脱管状态
55. person.setName("hmk2");
56. em.merge(person); //把脱管状态变为托管状态,merge可以自动选择insert or update 数据
57. em.getTransaction().commit();
58. em.close();
59. factory.close();
60. }
62. @Test public void remove(){
63. EntityManagerFactory factory = Persistence.createEntityManagerFactory("mysqlJPA");
64. EntityManager em = factory.createEntityManager();
65. em.getTransaction().begin();
66. Person person = em.find(Person.class, 1);
67. em.remove(person); //删除实体
68. em.getTransaction().commit();
69. em.close();
70. factory.close();
71. }
73. @Test public void find(){
74. EntityManagerFactory factory = Persistence.createEntityManagerFactory("mysqlJPA");
75. EntityManager em = factory.createEntityManager();
76. Person person = em.find(Person.class, 2); //类似于hibernate的get方法,没找到数据时，返回null
77. System.out.println(person.getName());
78. em.close();
79. factory.close();
80. }
81. @Test public void find2(){
82. EntityManagerFactory factory = Persistence.createEntityManagerFactory("mysqlJPA");
83. EntityManager em = factory.createEntityManager();
84. Person person = em.getReference(Person.class, 2); //类似于hibernate的load方法,延迟加载.没相应数据时会出现异常
85. System.out.println(person.getName()); //真正调用时才查找数据
86. em.close();
87. factory.close();
88. }
89. }

d）、运行junit测试代码里的相应方法就行可以。

## 持久性解决方案比较

|  |  |  |  |
| --- | --- | --- | --- |
| 特性 | iBATIS | Hibernate | JPA |
| 简易性 | 优 | 良 | 良 |
| 完整的ORM解决方案 | 一般 | 优 | 优 |
| 对数据模型改变的适应性 | 良 | 一般 | 一般 |
| 复杂性 | 优 | 一般 | 一般 |
| 对SQL的依赖 | 良 | 一般 | 一般 |
| 性能 | 优 | 优 | 不适用\* |
| 跨不同关系数据库的移植性 | 一般 | 优 | 不适用\* |
| 非Java平台的移植性 | 优 | 良 | 不支持 |
| 社区支持和文档 | 一般 | 良 | 良 |

\*JPA对这些特性的支持取决于持久性提供程序，最终的结果可能会视情况各异。