**Hibernate Annotations Mapping**

**What is Annotation?**

*Annotations*, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Annotations have a number of uses, among them:

* **Information for the compiler** — Annotations can be used by the compiler to detect errors or suppress warnings.
* **Compile-time and deployment-time processing** — Software tools can process annotation information to generate code, XML files, and so forth.
* **Runtime processing** — Some annotations are available to be examined at runtime.

**Advantages Hibernate Annotations:**

Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development.

* Compile-time checking : writing in Java (instead of Xml) is very user-friendly in the IDE nowadays. **No more typos** discovered when starting your application (incremental compilation), not that much to remember (**completion**)...
* Localized with the code (class level) : instead of having to open two files (java and xml) to get the full story, with one annotated java file, you **open only one file**. This is less repetitive, faster in the long run.
* Localized with the code (method or field level) : because the annotation go on a method (or field), there is **no need to specify the method** it belongs to. That redondant information is not given, which is shorter, and always coherent (even after a code refactoring for example). Maintenance is so much faster.
* **Tools** (javadoc, other tools using reflection) can use the annotations for some other requirements.
* Annotations are newer than the xml, the team used the input they had received at the time to provide **better default values**. Xml has some, but can't change much for compatibility reasons. Often, with the annotations technology, you **write no annotation at all, and it works**. Imagine the time-saving, especially during development.

**jars required for hibernate annotations**

* Anttr-2.7.6.jar
* asm.jar
* asm-attrs.jar
* cglib-2.1.3.jar
* commons-collections-2.1.1.jar
* commons-logging-1.0.4.jar

**Hibernate Annotation Example:**

**package** com.model;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.JoinColumn;

**import** javax.persistence.ManyToOne;

**import** javax.persistence.Table;

@Entity

@Table(name="user")

**public** **class** UserModel {

@Id

@GeneratedValue(strategy=GenerationType.*AUTO*)

@Column(name="user\_id")

**private** Integer userId;

@Column(name="name",unique=**true**)

**private** String name;

@ManyToOne

@JoinColumn(name="add\_id")

**private** Address address;

@ManyToOne

@JoinColumn(name="dept\_id")

**private** Department department;

**public** Department getDepatment() {

**return** department;

}

**public** **void** setDepatment(Department department) {

**this**.department = department;

}

**public** UserModel(){}

**public** Integer getUserId() {

**return** userId;

}

**public** **void** setUserId(Integer userId) {

**this**.userId = userId;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Address getAddress() {

**return** address;

}

**public** **void** setAddress(Address address) {

**this**.address = address;

}

}

**package** com.model;

**import** java.util.Set;

**import** javax.persistence.CascadeType;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.OneToMany;

**import** javax.persistence.Table;

@Entity

@Table(name="department")

**public** **class** Department {

@Id

@GeneratedValue(strategy=GenerationType.*AUTO*)

@Column(name="dept\_id")

**private** Integer deptId;

@Column(name="name")

**private** String deptName;

@OneToMany(mappedBy="department",cascade=CascadeType.*ALL* )

**private** Set<UserModel> user;

**public** Integer getDeptId() {

**return** deptId;

}

**public** **void** setDeptId(Integer deptId) {

**this**.deptId = deptId;

}

**public** String getDeptName() {

**return** deptName;

}

**public** **void** setDeptName(String deptName) {

**this**.deptName = deptName;

}

**public** Set<UserModel> getUser() {

**return** user;

}

**public** **void** setUser(Set<UserModel> user) {

**this**.user = user;

}

}

**package** com.model;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name="address")

**public** **class** Address {

@Id

@GeneratedValue(strategy=GenerationType.*AUTO*)

@Column(name="add\_id")

**private** **int** addId;

@Column(name="country")

**private** String country;

@Column(name="city")

**private** String city;

**public** String getCountry() {

**return** country;

}

**public** **void** setCountry(String country) {

**this**.country = country;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

**public** **int** getAddId() {

**return** addId;

}

**public** **void** setAddId(**int** addId) {

**this**.addId = addId;

}

}

**package** com.connection;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.cfg.AnnotationConfiguration;

**import** org.hibernate.cfg.Configuration;

**public** **class** Connection {

**private** **static** SessionFactory *sessionFactory*=**null**;

**private** Connection(){}

**public** **static** SessionFactory getSessionFactory(){

**if**(*sessionFactory*==**null**){

*sessionFactory*=**new** AnnotationConfiguration().configure().buildSessionFactory();

}

**return** *sessionFactory*;

}

}

**package** com.dao;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.Set;

**import** org.hibernate.Query;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** com.model.Address;

**import** com.model.Author;

**import** com.model.Book;

**import** com.model.Department;

**import** com.model.UserModel;

**public** **class** UserDao {

**public** **static** SessionFactory *factory* = com.connection.Connection.*getSessionFactory*();

Session session = *factory*.openSession();

Transaction tr = session.beginTransaction();

**public** **void** addUser(String std\_name){

UserModel model=**new** UserModel();

model.setName(std\_name);

String hql="from Address where addId ="+1;

Query query=session.createQuery(hql);

String hql2="from Department where deptId ="+1;

Query query2=session.createQuery(hql2);

Address address=(Address) query.uniqueResult();

Department dept=(Department) query2.uniqueResult();

model.setAddress(address);

model.setDepatment(dept);;

session.persist(model);

tr.commit();

System.*out*.println("Record add");

}

**public** **void** deleteUser(**int** id){

String hql="from UserModel where userId =:userId";

Query query=session.createQuery(hql);

query.setParameter("userId", id);

UserModel model=(UserModel) query.uniqueResult();

session.delete(model);

tr.commit();

System.*out*.println("Record delete");

}

**public** **void** addAddress(String country,String city){

Address addres=**new** Address();

addres.setCountry(country);

addres.setCity(city);;

session.persist(addres);

tr.commit();

System.*out*.println("Record add");

}

**public** **void** addDepartment(String name){

Set <UserModel> useres=**new** HashSet<UserModel>();

UserModel user1=**new** UserModel();

Department dept=**new** Department();

dept.setDeptName(name);

String hql="from Address where addId ="+2;

Query query=session.createQuery(hql);

Address address=(Address) query.uniqueResult();

user1.setName("a");

user1.setDepatment(dept);

user1.setAddress(address);

UserModel user2=**new** UserModel();

user2.setName("b");

user2.setDepatment(dept);

user2.setAddress(address);

useres.add(user1);

useres.add(user2);

dept.setUser(useres);

session.save(dept);

tr.commit();

System.*out*.println("Record add");

}

**public** **static** **void** main (String arg[]){

UserDao dao=**new** UserDao();

//dao.addUser("Adeel22222");

//dao.delete(2);

//dao.addAddress("pakistan", "piaro");

dao.addDepartment("BCS");

}

}

**hibernate.cfg.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

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

<property name=*"hibernate.connection.password"*>123</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/userDb?createDatabaseIfNotExist=true</property>

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

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>

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

<property name=*"hibernate.show\_sql"*>true</property>

<mapping class=*"com.model.UserModel"*></mapping>

<mapping class=*"com.model.Address"*/>

<mapping class=*"com.model.Department"*/>

</session-factory>

</hibernate-configuration>

**Hibernate Annotations**

@Entity Annotation:

Contained in the **javax.persistence**package, so we import this package as the first step. Second we used the**@Entity** annotation to the class which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

## @Table Annotation:

The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.

The @Table annotation provides four attributes, allowing you to override the name of the table, its catalogue, and its schema, and enforce unique constraints on columns in the table. Now we are using just table name

## @Id and @GeneratedValue Annotations:

Each entity bean will have a primary key, which you annotate on the class with the **@Id** annotation. The primary key can be a single field or a combination of multiple fields depending on your table structure.

By default, the @Id annotation will automatically determine the most appropriate primary key generation strategy to be used but you can override this by applying the **@GeneratedValue** annotation which takes two parameters **strategy** and **generator**

## @Column Annotation:

The @Column annotation is used to specify the details of the column to which a field or property will be mapped. You can use column annotation with the following most commonly used attributes:

* **name** attribute permits the name of the column to be explicitly specified.
* **length** attribute permits the size of the column used to map a value particularly for a String value.
* **nullable** attribute permits the column to be marked NOT NULL when the schema is generated.
* **unique** attribute permits the column to be marked as containing only unique values.

## @OneToMany Relationship and ManyToOne:

The term “bidirectional” literally means “functioning in two directions”, which is the concept that we will apply in our relationships between two Java objects. When we have a bidirectional relationship between objects, it means that we are able to access Object A from Object B, and Object B from Object A.

One **critical** thing to remember here is that there’s a difference between a reference from the One-to-Many side and the Many-to-One side.

When you traverse from the “Many” side to the “One” side, you only need to make reference to one object, which is why the UserModel class holds a single reference to an Department class via the private Department  department  instance variable.

However, when you traverse from the “One” to the “Many” side, you will need to hold a reference to MANY objects.

Does that make sense? Many-to-One equals one reference, One-to-Many equals many references.

So what will this look like? Well, it means that the Department will need to hold many references to its Users, and we accomplish this by storing them as a Collection.

In this example I will use a Set of UserModel objects in my Department class file

**@joincolumn:**

The annotation @JoinColumn indicates that this entity is the owner of the relationship (that is: the corresponding table has a column with a foreign key to the referenced table)

The annotation **@JoinColumn** is used to specify a mapped column for joining an entity association, this annotation can be used in any class (Parent or Child) but it should ideally be used only in one side (either in parent class or in Child class not in both) here in this case i used it in the Child side (Branch class) of the bi directional relationship indicating the foreign key in the Branch class.

**mapped by attribute:**

Using "**mappedBy**" attribute of mapping annotations(like @OneToOne, @OneToMany, @ManyToMany) for bi-directional relationship. This attribute allows you to refer the associated entities from both sides. If "X" has association with "Y" then you can get X from Y and Y from X.