主键生成策略

ORM映射的基本原理

- ✓ 1. 基于相同实体的类和表,实现相互映射,类的对象对应了表中的记录,不同对象对应不同的记录,不同的记录对应不同的对象.
- ✓ 2. 表中不同的记录通过主键来区分,不同的对象通过对象id来区分,对象id是对象中的一个成员变量,该变量的值唯一性的标识了对象.
- ✓ 3. 对象id和表主键的映射构成了ORM的核心
- ✓ 4. 数据库往往具备特定的主键生成算法,而对象系统则不具备,所以要配置特定的策略,以使对象具备和数据库中的数据同步的对象标识.

配置主键生成策略

- ✓ 主键生成策略是通过配置映射文件实现的
- ✓ 在映射文件中<id>标签对应的是表中的主键列
- ✓ <generator>子标签配置主键的生成策略
- ✓ class属性值指定具体的策略，可以是以下值：
  - ➢ increment、identity、sequence、hilo、seqhilo、uuid、native、assigned

主键映射的基本语法

```
<id
    name=“属性名称"
    type=“属性类型"
    column=“列名称">
    <generator class=“生成策略名称"/>
</id>
```

创建实验表product产品表

```
create table product
(
id number(6),
prodname varchar2(100)
);
```

assigned主键策略

- ✓ 该种主键策略下:
  - ➢ 由程序设置主键.
  - ➢ 设置必须在save操作之前完成.
  - ➢ ORM框架和数据库都是被动的主键值保存者和传递者
- ✓ Asinged策略配置方式
  - ➢ 这种类型的主键可以是任何类型的字段
  - ➢ Class属性是该主键生成算法对应的类的名字,默认在id包中

assigned主键策略示例

```
<generator class="assigned"/>
```

```
Product prod = new Product();
prod.setId(10);
prod.setProdname("苹果");
session.save(prod);
tx.commit();
```

increment主键策略

- ✓ 该种主键策略下:
  - ➢ 由Hibernate维护的主键生成策略
  - ➢ 其算法为将数据库中最大的Id记录增加1,作为新对象的id
- ✓ Increment策略配置方式
  - ➢ 这种类型的主键只可以整型字段
  - ➢ Class属性是该主键生成算法对应的类的名字
  - ➢ 在并发性很高的程序中,容易引起主键的重复

increment主键策略示例

```
<generator class="increment"/>
```

identity主键策略

- ✓ 该种主键策略下:
  - ➢ 由底层数据库维护的主键生成策略
  - ➢ 需要特定数据库特性支持
  - ➢ Mysql 数据库的auto_increment数据类型
  - ➢ Sql Server数据库的identity数据类型
  - ➢ Oracle没有对应的数据类型，用不了
- ✓ Identity策略配置方式
  - ➢ 这种类型的主键只可以整型字段
  - ➢ Class属性是该主键生成算法对应的类的名字
  - ➢ 只能在单个表的范围保证数据的唯一性

identity主键策略示例

```
<generator class="identity"/>
```

sequence主键策略

- ✓ 该种主键策略下:
  - ➢ 由底层数据库维护的主键生成策略
  - ➢ 需要特定数据库特性支持
  - ➢ Oracle数据库需要创建一个序列对象
- ✓ Sequence策略配置方式
  - ➢ 这种类型的主键只可以整型字段
  - ➢ Class属性是该主键生成算法对应的类的名字
  - ➢ 只能在单个数据库范围内保证数据的唯一性

sequence主键策略示例

创建序列

```
create sequence prod_seq start with 100 increment by 2;
```

```
<generator class="sequence">
    <param name="sequence">prod_seq</param>
</generator>
```

native主键策略略

- 该种主键策略下:
  - 根据底层数据库的特性,自动选择Identity, Sequence或Hilo主键生成方式
  - Oracle数据库必须提供一个名称为hibernate_squence的序列
- Native策略配置方式
  - 这种类型的主键只可以整型字段
  - Class属性是该主键生成算法对应的类的名字

native主键策略示例

```
<generator class="native"/>
```

Hilo高位低位主键策略

- ✓ 该种主键策略下:
  - ➢ 借助数据库中的辅助表生成整型主键.
  - ➢ 该主键由ORM框架生成.
- ✓ Hilo策略配置方式
  - ➢ 这种类型的主键只可以整型字段
  - ➢ Class属性是该主键生成算法对应的类的名字

Hilo策略辅助表

- ✓ 辅助表结构:
  - ➢ 辅助表中至少具备一个int(10)型的字段,且具有一条记录,该记录的值不为空.
  - ➢ 表的名称和字段名随意
- ✓ 默认结构
  - ➢ 表的名称为my_unique_key
  - ➢ 字段的默认名称为next_hi

Hilo高位低位主键策略示例

```
//创建一个辅助表做算法种子值
create table my_hilo_key
(
next_value number(10) primary key
);
//插入一条记录
insert into my_hilo_key values (200);

 <generator class="hilo">
      <param name="table">my_hilo_key</param>
      <param name="column">next_value</param>
 </generator>
```

UUID主键策略

- ✓ 该种主键策略下:
  - ➢ uuid为一种全局唯一标识符号
  - ➢ 能够保证多并行系统之间主键的唯一性
  - ➢ 主键宽度为32个字符
  - ➢ UUID包含：IP地址，JVM的启动时间（精确到0.25秒），系统时间和一个计数器值（在JVM中唯一）
- ✓ UUID策略配置方式
  - ➢ 这种类型的主键为字符串字段
  - ➢ Class属性是该主键生成算法对应的类的名字

UUID主键策略示例

```
<generator class="uuid"/>
```

自然主键和代理主键

- ✓ 自然主键就是主键列本身有特定逻辑意义,和实际现实相联系,比较直观，例如用登陆用户名作主键，身份证号码做主键，银行账号做主键都是自然主键
- ✓ 代理主键是通常的整型ID, 除了具有标识意义外,没有其它逻辑意义，一般就是一个自动编号，起到唯一标识的作用
- ✓ 多数情况下推荐使用代理主键

单列自然主键

- ✓ 主键需要设置为Assigned方式;
- ✓ 只有在Assigned主键模式下,主键列的set方法才起作用.
- ✓ 其他策略都是自动生成主键值，无法手动给主键赋上有实际意义的值

复合自然主键

- ✓ 使用两个或两个以上的列作为表中的主键即为复合主键
- ✓ 映射复合自然主键的两种方法
  - ➢ 直接使用映射文件配置
  - ➢ 创建一个辅助的复合主键类

创建实验表score成绩表

- ✓ 例如：学号，课程号，考试成绩三列组成一个成绩表，其中学号和课程号就是复合主键
- ✓ **create table** score
  (
  sid **number**(6),
  cid **number**(6),
  result number(3),
  constraint PK_SID_CID **primary key** (sid,cid)
  );

直接使用映射文件配置

- ✓ 1. 持久化类要实现如下条件
  - ➢ 实现java.io.Serializable接口
  - ➢ 重写equals方法.
  - ➢ 重写hashCode方法.
- ✓ 2.使用<composite-id>进行映射

实体类代码示例

```
public class Score implements java.io.Serializable {
    private Integer sid;
    private Integer cid;
    private Integer result;

    //GET和SET略

    //覆盖equals方法
    public boolean equals(Object obj) {
        if (obj == null)
            return false;
        if (this == obj)
            return true;
        if (this.getClass() != obj.getClass())
            return false;
        Score other = (Score) obj;
        if (sid != null && cid != null && sid.equals(other.sid)
                && cid.equals(other.cid)) {
            return true;
        }
        return false;
    }

    //覆盖hashCode方法
    public int hashCode(){
        int result = 17;

        result = result * 37 +
    sid.hashCode() + cid.hashCode();

        return result;
    }
}
```

映射文件示例

```
<hibernate-mapping>
    <class name="com.Score" table="SCORE">
        <composite-id>
            <key-property name="sid" type="java.lang.Integer"/>
            <key-property name="cid" type="java.lang.Integer"/>
        </composite-id>
        <property name="result" type="java.lang.Integer"/>
    </class>
</hibernate-mapping>
```

测试类示例

```
public class TestScore {
    public static void main(String[] args) {
        Session session = HibernateSessionFactory.getSession();
        Transaction tx = session.beginTransaction();
        try {
            Score score = new Score();
            score.setSid(101);//设置学号
            score.setCid(2);//设置课号
            score.setResult(59);

            session.save(score);

            tx.commit();
        }catch (Exception e){
            e.printStackTrace();
            tx.rollback();
        }finally {
            session.close();
        }
    }
}
```

创建一个辅助的复合主键类实现映射

- ✓ 复合主键类
  - ➢ 实现java.io.Serializable接口
  - ➢ 包含主键中的各个字段
  - ➢ 重写equals方法.
  - ➢ 重写hashCode方法.
- ✓ 持久化实体类
  - ➢ 使用复合主键类作为自己的成员变量,对应主键
  - ➢ 使用<composite-id>进行映射

复合主键类代码示例

```
package com;
public class ScoreId implements
    java.io.Serializable {

    private Integer sid;
    private Integer cid;

    //GET和SET略

    //覆盖hashCode方法
    public int hashCode(){
            int result = 17;
           result = result * 37 + sid.hashCode()
    + cid.hashCode();
            return result;
    }

    //覆盖equals方法
        public boolean equals(Object obj) {
            if (obj == null)
                    return false;
            if (this == obj)
                    return true;
            if (this.getClass() != obj.getClass())
                    return false;
            ScoreId other = (ScoreId) obj;
            if (sid != null && cid != null && sid.equals(other.sid)
                                    && cid.equals(other.cid)) {
                    return true;
            }
            return false;
        }
}
```

实体类示例代码

```
package com;
public class Score  {
	private ScoreId scoreId;
	private Integer result;

	//GET和SET略
}
```

映射文件示例代码

```
<hibernate-mapping>
    <class name="com.Score" table="SCORE">
        <composite-id name="scoreId" class="com.ScoreId">
            <key-property name="sid" type="java.lang.Integer"/>
            <key-property name="cid" type="java.lang.Integer"/>
        </composite-id>
        <property name="result" column="result" type="java.lang.Integer"/>
    </class>
</hibernate-mapping>
```

测试类示例代码

```
public class TestScore {
    public static void main(String[] args) {
        Session session = HibernateSessionFactory.getSession();
        Transaction tx = session.beginTransaction();
        try {
            //创建一个复合主键类对象，给两个主键赋值
            ScoreId scoreId = new ScoreId();
            scoreId.setSid(200);
            scoreId.setCid(3);
            //创建持久化类对象
            Score score = new Score();
            score.setScoreId(scoreId);
            score.setResult(59);
            session.save(score);
            tx.commit();
        }catch (Exception e){
            e.printStackTrace();
            tx.rollback();
        }finally {
            session.close();
        }
    }
}
```