# Hibernate

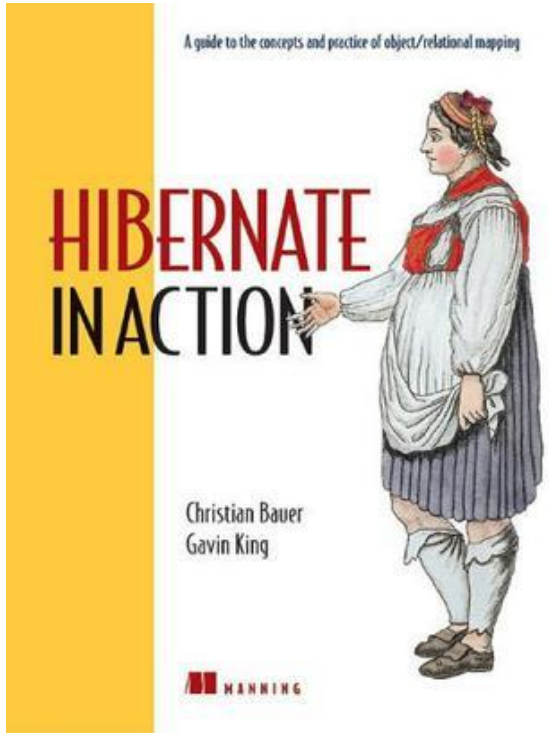
4.x

凯盛软件

# 推荐书籍

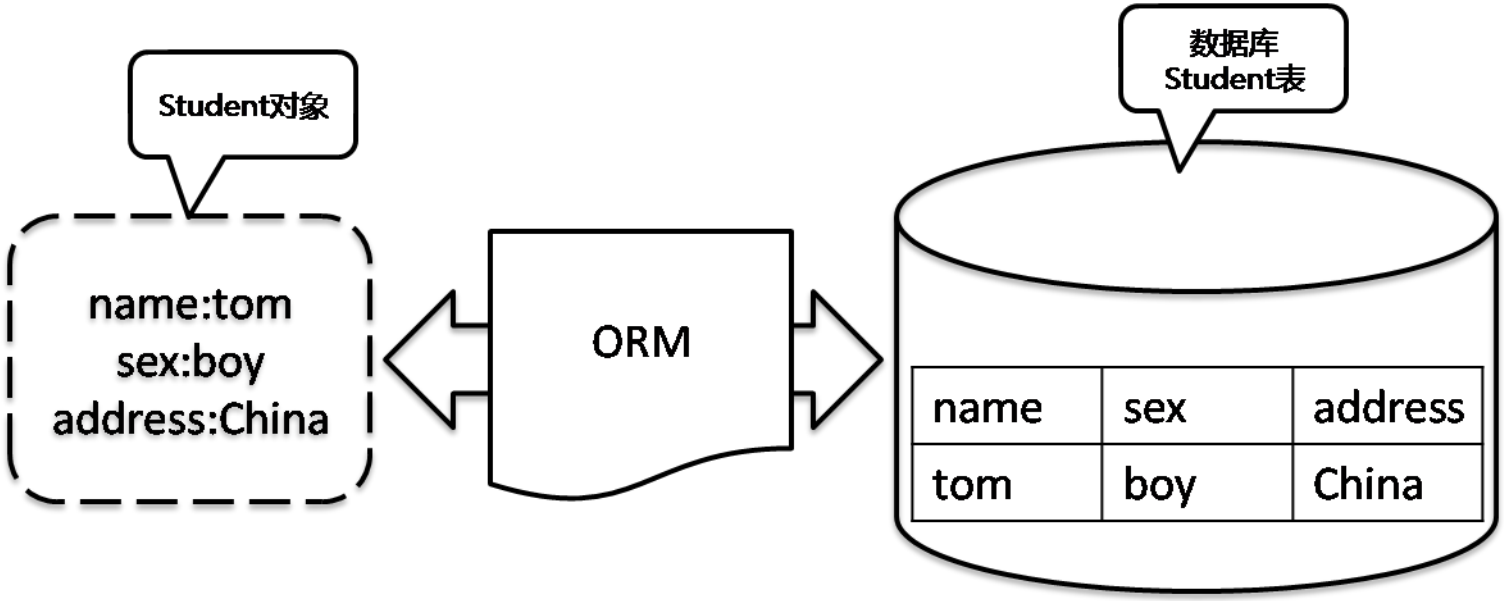
凯盛软件


Broadview
深入浅出
Hibernate
夏昕 曹晓钢 唐勇 编著
夏昕
曹晓钢
唐勇
電子工業出版社


A guide to the concepts and practice of object/relational mapping
HIBERNATE
IN ACTION
Christian Bauer
Gavin King
MANNING

# ORM

凯盛软件


Student对象
name:tom
sex:boy
address:China
ORM
数据库
Student表
name
sex
address
tom
boy
China

# ORM框架

凯盛软件


HIBERNATE


OpenJPA


mybatis
NEW HOME OF THE WORLD'S MOST POPULAR SQL MAPPING FRAMEWORK.

# ORM为我们做什么

凯盛软件

减少乏味的代码

更加面向对象的设计

更好的性能

更好的移植性

# 使用Hibernate

凯盛软件

http://hibernate.org/

1. 加入Hibernate的jar包
2. 创建持久化类（pojo）
3. 创建映射文件（xxx.hbm.xml）
4. 创建Hibernate配置文件（hibernate.cfg.xml）
5. 运行

# POJO（持久化类）

凯盛软件

pojo:**P**lain **O**rdinary **J**ava **O**bject(无格式的Java对象)

Hibernate对pojo的要求：

- 属性要有对应的get和set方法
- 要有无参数的默认构造方法
- 不要使用final进行修饰

# 映射文件

凯盛软件


name:tom
sex:boy
address:China
映射文件
name
sex
address
tom
boy
China

# 映射文件示例

凯盛软件

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
        <class table="t_user" name="com.kaishengit.entity.User">
                <id name="id" column="id">
                        <generator class="native"/>
                </id>
                <property name="username"/>
                <property name="password" column="password"/>
        </class>
</hibernate-mapping>
```

# Hibernate配置文件

凯盛软件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
                <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
                <property   name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
                <property name="hibernate.connection.url">jdbc:mysql:///mydb</property>
                <property name="hibernate.connection.username">root</property>
                <property name="hibernate.connection.password"></property>
                <property name="hibernate.current_session_context_class">thread</property>
                <property name="hibernate.show_sql">true</property>
                <property name="hibernate.connection.pool_size">1</property>

                <mapping resource="com/kaishengit/entity/User.hbm.xml"/>
        </session-factory>
</hibernate-configuration>
```

# 使用Hibernate

凯盛软件

```
Configuration cfg = new Configuration().configure();

ServiceRegistry serviceRegistry = new ServiceRegistryBuilder().applySettings(cfg.getProperties()).buildServiceRegistry();


SessionFactory factory = cfg.buildSessionFactory(serviceRegistry);


Session session = factory.getCurrentSession();

session.beginTransaction();


//code…


session.getTransaction().commit();
```

# 保存一个对象

凯盛软件

```
session.beginTransaction();

User user = new User();
user.setUsername("aa");
user.setPassword("123");

session.save(user);

session.getTransaction().commit();
```

# 根据主键查找对象

凯盛软件

```
session.beginTransaction();

User user = (User) session.get(User.class, 1);
System.out.println(user.getUsername());

session.getTransaction().commit();
```

# 修改对象

凯盛软件

```
Session session = factory.getCurrentSession();
session.beginTransaction();

User user = (User) session.get(User.class, 1);
user.setUsername("Alex");

session.getTransaction().commit();
```

# 删除对象

凯盛软件

```
Session session = factory.getCurrentSession();
session.beginTransaction();

User user = (User) session.get(User.class, 1);
session.delete(user);

session.getTransaction().commit();
```

# 查找所有对象

凯盛软件

```
Session session = factory.getCurrentSession();
session.beginTransaction();

Query query = session.createQuery("from User");
List<User> userList = query.list();
System.out.println(userList.size());

session.getTransaction().commit();
```

# HibernateUtil

凯盛软件

```
public class HibernateUtil {

    private HibernateUtil(){}

    public static SessionFactory factory = builderSessionFactory();

    private static SessionFactory builderSessionFactory() {
            Configuration cfg = new Configuration().configure();
            ServiceRegistry serviceRegistry = new
            ServiceRegistryBuilder().applySettings(cfg.getProperties()).buildServiceRegistry();
            SessionFactory factory = cfg.buildSessionFactory(serviceRegistry);
            return factory;
    }

    public static Session getSession() {
            return factory.getCurrentSession();
    }
}
```

# 持久化对象的生命周期

凯盛软件

瞬态(自由态)

持久态

托管态(游离态)

# 自由态

凯盛软件

持久化对象的自由态，指的是对象在内存中存在，但是在数据库中并没有数据与其关联。比如Student student = new Student()，这里的student对象就是一个自由态的持久化对象。

# 持久态

凯盛软件

持久态指的是持久化对象处于由Hibernate管理的状态，这种状态下持久化对象的变化将会被同步到数据库中

# 游离态

凯盛软件

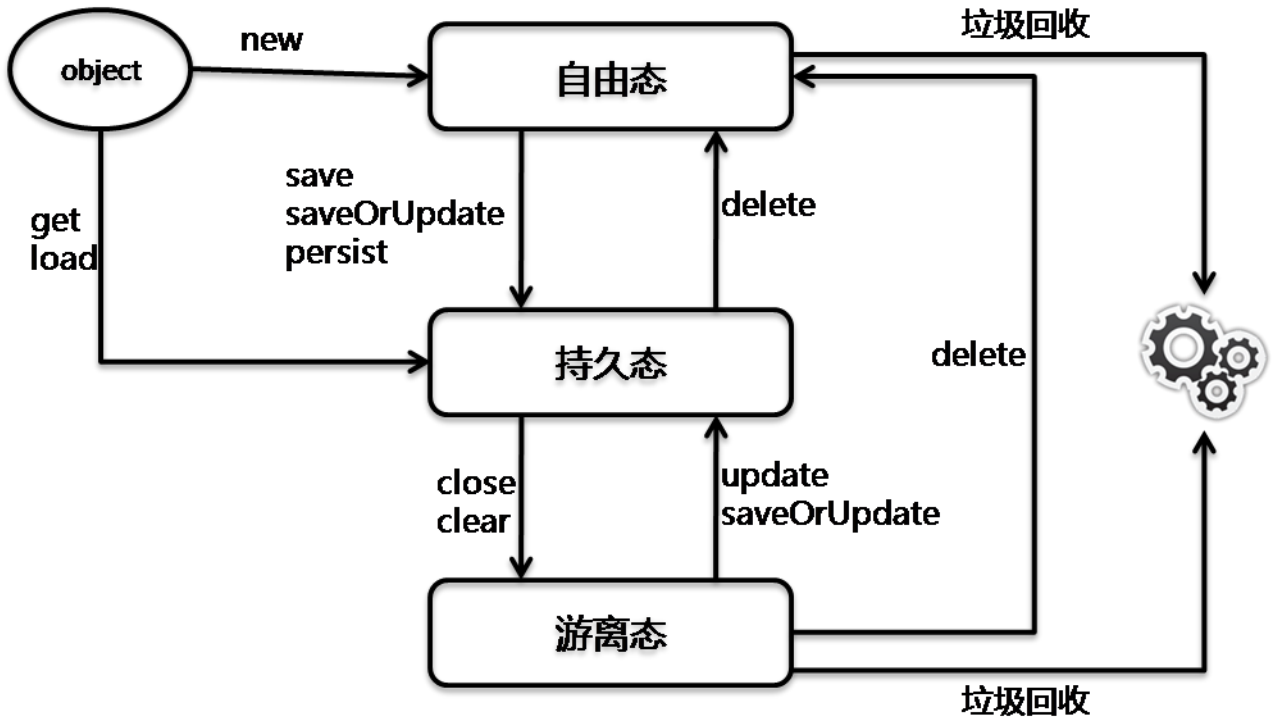
处于持久态的对象，在其对应的Session实例关闭后，此时对象进入游离态。也就是说Session实例是持久态对象的宿主环境，一旦宿主环境失效，那么持久态对象进入游离状态

游离态和自由态的区别：

- 游离态对象可以再次与Session进行关联而成为持久态对象。
- 自由态对象在数据库中没有数据与其对应，但是游离态对象在数据库中有数据与其对应，只不过当前对象不在Session环境中而已。从对象的是否有主键值可以做简单的判断。

# 三种状态的转换

凯盛软件



# load和get方法

凯盛软件

- get和load方法都是利用对象的主键值获取相应的对象，并可以使对象处于持久状态。
- load方法获取对象时不会立即执行查询操作，而是在第一次使用对象是再去执行查询操作。如果查询的对象在数据库中不存在，load方法返回值不会为null，在第一次使用时抛出org.hibernate.ObjectNotFoundException异常。
- 使用get方法获取对象时会立即执行查询操作，并且对象在数据库中不存在时返回null值。

# save和persist方法

凯盛软件

- save和persist方法都是将持久化对象保存到数据库中
- sava方法成功执行后，返回持久化对象的ID
- persist方法成功执行后，不会返回持久化对象的ID，persist方法是JPA中推荐使用的方法

# save和update方法

凯盛软件

- save方法是将自由态的对象进行保存
- update方法是将游离态的对象进行保存

# update和saveOrUpdate方法

凯盛软件

- 如果一个对象是游离态或持久态，对其执行update方法后会将对象的修改同步到数据库中，如果该对象是自由态，则执行update方法是没有作用的

- 在执行saveOrUpdate方法时该方法会自动判断对象的状态，如果为自由态则执行save操作，如果为游离态或持久态则执行update操作

# update和merge方法

凯盛软件

- 如果持久化对象在数据库中存在，使用merge操作时进行同步操作。如果对象在数据库不存在，merge对象则进行保存操作

- 如果对象是游离状态，经过update操作后，对象转换为持久态。但是经过merge操作后，对象状态依然是游离态

# saveOrUpdate和merge方法

凯盛软件

saveOrUpdate方法和merge方法的区别在于如果session中存在两个主键值相同的对象，进行saveOrUpdate操作时会有异常抛出。这时必须使用merge进行操作。

```
session.beginTransaction();
User user = new User();
user.setId(120);
user.setUserName("aaaaaaaa");
user.setUserPwd("123123");

User user2 = (User) session.get(User.class, 120);

session.saveOrUpdate(user);//ERROR
session.getTransaction().commit();
```

# clear方法和flush方法

凯盛软件

clear方法是将Session中对象全部清除，当前在Session中的对象由持久态转换为游离态。flush方法则是将持久态对象的更改同步到数据库中。

# HQL

凯盛软件

HQL（Hibernate Query Language）提供了丰富灵活的查询方式，使用HQL进行查询也是Hibernate官方推荐使用的查询方式。

HQL在语法结构上和SQL语句十分的相同，所以可以很快的上手进行使用。使用HQL需要用到Hibernate中的Query对象，该对象专门执行HQL方式的操作。

# 查找所有

凯盛软件

```
session.beginTransaction();
String hql = "from User";
Query query = session.createQuery(hql);
List<User> userList = query.list();
for(User user:userList){
        System.out.println(user.getUserName());
}
session.getTransaction().commit();
```

# where

凯盛软件

```
session.beginTransaction();
String hql = "from User where userName = 'James'";
Query query = session.createQuery(hql);
List<User> userList = query.list();
for(User user:userList){
        System.out.println(user.getUserName());
}
session.getTransaction().commit();
```

在HQL中where语句中使用的是持久化对象的属性名，比如上面示例中的userName。当然在HQL中也可以使用别名：

```
String hql = "from User as u where u.userName = 'James'";
```

# 过滤条件

凯盛软件

在where语句中还可以使用各种过滤条件，如：=、<>、<、>、>=、<=、between、not between、in、not in、is、like、and、or等。

- from Student where age > 20;
- from Student where age between 20 and 30;
- from Student where name is null;
- from Student where name like ‘小%’;
- from Student where name like ‘小%’ and age < 30

# 获取一个不完整对象

凯盛软件

```
session.beginTransaction();

String hql = "select userName from User";

Query query = session.createQuery(hql);

List nameList = query.list();

for(Object obj:nameList){

        System.out.println(obj);

}

session.getTransaction().commit();
```

```
session.beginTransaction();
String hql = "select userName,userPwd from User";
Query query = session.createQuery(hql);
List nameList = query.list();
for(Object obj:nameList){
        Object[] array = (Object[]) obj;
        System.out.println("name:" + array[0]);
        System.out.println("pwd:" + array[1]);
}
session.getTransaction().commit();
```

# 统计和分组查询

凯盛软件

```
session.beginTransaction();
String hql = "select count(*),max(id) from User";
Query query = session.createQuery(hql);
List nameList = query.list();
for(Object obj:nameList){
        Object[] array = (Object[]) obj;
        System.out.println("count:" + array[0]);
        System.out.println("max:" + array[1]);
}
session.getTransaction().commit();
```

# 更多写法...

凯盛软件

- select distinct name from Student;
- select max(age) from Student;
- select count(age),age from Student group by age;
- from Student order by age;

# HQL占位符

凯盛软件

```
session.beginTransaction();
String hql = "from User where userName = ?";
Query query = session.createQuery(hql);

query.setString(0, "James");

List<User> userList = query.list();

for(User user:userList){
        System.out.println(user.getUserName());
}
session.getTransaction().commit();
```

# HQL引用占位符

凯盛软件

```
session.beginTransaction();
String hql = "from User where userName = :name";
Query query = session.createQuery(hql);

query.setParameter("name", "James");

List<User> userList = query.list();

for(User user:userList){
        System.out.println(user.getUserName());
}

session.getTransaction().commit();
```

# HQL分页

凯盛软件

```
session.beginTransaction();
String hql = "from User";
Query query = session.createQuery(hql);

query.setFirstResult(0);
query.setMaxResults(2);

List<User> userList = query.list();
for(User user:userList){
        System.out.println(user.getUserName());
}
session.getTransaction().commit();
```

# Criteria查询

凯盛软件

Criteria对象提供了一种面向对象的方式查询数据库。Criteria对象需要使用Session对象来获得

一个Criteria对象表示对一个持久化类的查询

# 查询所有

凯盛软件

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);

List<User> userList = c.list();

for(User user:userList){
        System.out.println(user.getUserName());
}

session.getTransaction().commit();
```

# Where

凯盛软件

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);

c.add(Restrictions.eq("userName", "James"));

List<User> userList = c.list();
for(User user:userList){
        System.out.println(user.getUserName());
}
session.getTransaction().commit();
```

# Restrictions对象

凯盛软件

| 方法名称 | 对应SQL中的表达式 |
| --- | --- |
| Restrictions.eq | field = value |
| Restrictions.gt | field > value |
| Restrictions.lt | field < value |
| Restrictions.ge | field >= value |
| Restrictions.le | field <= value |
| Restrictions.between | field between value1 and value2 |
| Restrictions.in | field in(…) |
| Restrictions.and | and |
| Restrictions.or | or |
| Restrictions.like | field like value |

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);

c.add(Restrictions.like("userName", "J"));
c.add(Restrictions.eq("id", 120));

List<User> userList = c.list();
for(User user:userList){
        System.out.println(user.getUserName());
}
session.getTransaction().commit();
```

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);

c.add(Restrictions.or(Restrictions.eq("userName", "James"),
                Restrictions.eq("userName", "Alex")));

List<User> userList = c.list();
for(User user:userList){
System.out.println(user.getUserName());
}
session.getTransaction().commit();
```

# 获取唯一的记录

凯盛软件

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);
c.add(Restrictions.eq("id", 120));

User user = (User) c.uniqueResult();

System.out.println(user.getUserName());

session.getTransaction().commit();
```

# 分页

凯盛软件

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);

c.setFirstResult(0);
c.setMaxResults(5);

List<User> userList = c.list();

for(User user:userList){
        System.out.println(user.getUserName());
}

session.getTransaction().commit();
```

# 分组与统计

凯盛软件

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);

c.setProjection(Projections.sum("id"));

Object obj = c.uniqueResult();
System.out.println(obj);

session.getTransaction().commit();
```

# Projections对象

凯盛软件

| 方法名称 | 描述 |
| --- | --- |
| Projections.sum | 等于SQL中聚合函数sum |
| Projections.avg | 等于SQL中聚合函数avg |
| Projections.count | 等于SQL中聚合函数count |
| Projections .distinct | 去除重复记录 |
| Projections.max | 等于SQL中聚合函数max |
| Projections.min | 等于SQL中聚合函数min |
| Projections .groupProperty | 对指定的属性进行分组查询 |

# 多个统计与分组

凯盛软件

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);

ProjectionList projectionList = Projections.projectionList();
projectionList.add(Projections.sum("id"));
projectionList.add(Projections.min("id"));

c.setProjection(projectionList);

Object[] obj = (Object[]) c.uniqueResult();

System.out.println("sum:" + obj[0]);
System.out.println("min:" + obj[1]);


session.getTransaction().commit();
```

# 排序

凯盛软件

```
session.beginTransaction();

Criteria c = session.createCriteria(User.class);

c.addOrder(Order.desc("id"));

List<User> list = c.list();

for(User user : list){
        System.out.println(user.getUserName());
}


session.getTransaction().commit();
```

# 使用原生SQL查询

凯盛软件

```
session.beginTransaction();

String sql = "select id,username,userpwd from t_user";
List list = session.createSQLQuery(sql).list();

for(Object item : list){
        Object[] rows = (Object[]) item;
        System.out.println("id:" + rows[0] + "username:"
                + rows[1] + "userpwd:" + rows[2]);
}

session.getTransaction().commit();
```

```
session.beginTransaction();

String sql = "select id,username,userpwd from t_user";
SQLQuery query = session.createSQLQuery(sql).addEntity(User.class);

List<User> list = query.list();

for(User user : list){
        System.out.println(user.getUserName());
}

session.getTransaction().commit();
```

```
session.beginTransaction();

String sql = "select id,username,userpwd from t_user where id = 2";
SQLQuery query = session.createSQLQuery(sql).addEntity(User.class);

User user = (User) query.uniqueResult();

System.out.println(user.getUserName());

session.getTransaction().commit();
```

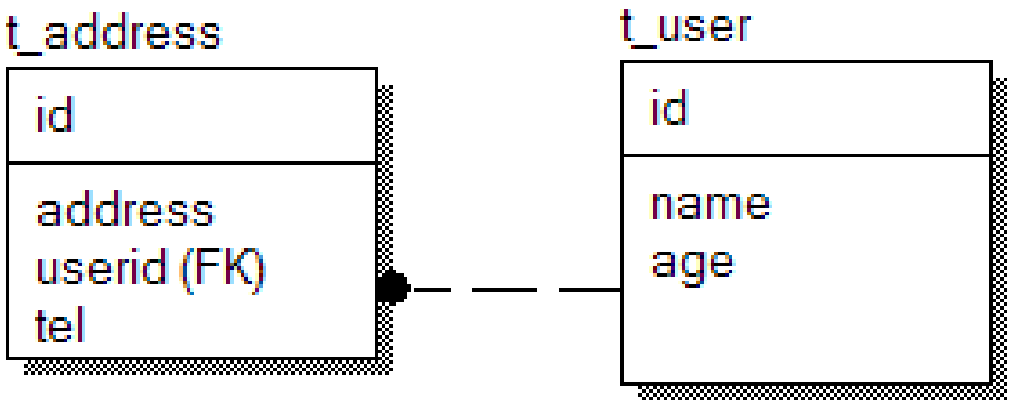
# 数据关联映射

凯盛软件

- 一对一
- **一对多**
- **多对多**

# 一对多 多对一

凯盛软件



# 持久化类设计

凯盛软件

```
public class User {

private int id;
private String name;
private int age;
private Set<Address> addressSet;

//get set method
}
```

```
public class Address {

private int id;
private String address;
private int tel;
private User user;

//get set method
}
```

# User.hbm.xml

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">

    <class name="User" table="t_user">
        <id name="id" column="id">
            <generator class="native"></generator>
        </id>
        <property name="name" column="name"></property>
        <property name="age" column="age"></property>

        <set name="addressSet">
            <key column="userid"/>
            <one-to-many class="Address"/>
        </set>
    </class>

</hibernate-mapping>
```

# Address.hbm.xml

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">

    <class name="Address" table="t_address">
        <id name="id">
                <generator class="native"/>
        </id>
        <property name="address"></property>
        <property name="tel"></property>
        <many-to-one name="user" class="User" column="userid"/>
    </class>

</hibernate-mapping>
```

# 放弃关系维护

凯盛软件

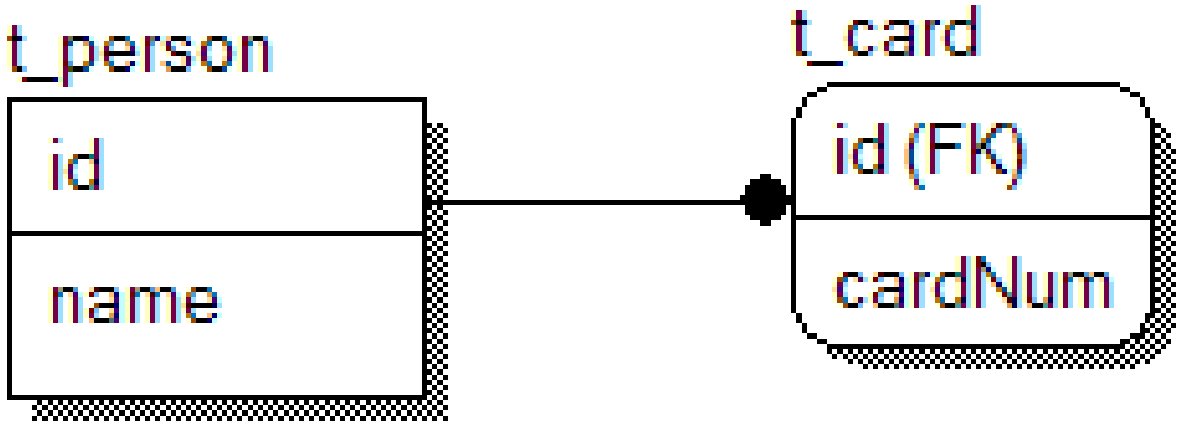
```
<hibernate-mapping package="com.kaishengit.pojo">
    <class name="User" table="t_user">
        <id name="id" column="id">
            <generator class="native"></generator>
        </id>
        <property name="name" column="name"></property>
        <property name="age" column="age"></property>

        <set name="addressSet" inverse="true">
            <key column="userid"/>
            <one-to-many class="Address"/>
        </set>
    </class>

</hibernate-mapping>
```

# 一对一

凯盛软件



# 持久化类设计

凯盛软件

```
public class Person {

    private int id;
    private String userName;
    private Card card;

    //get set Method
}
```

```
public class Card {

    private int id;
    private String cardNum;
    private Person person;

    //get set Method
}
```

# Person.hbm.xml

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">

        <class name="Person" table="t_person">
                <id name="id">
                    <generator class="native"></generator>
                </id>
            <property name="userName"></property>
            <one-to-one name="card" class="Card"></one-to-one>
        </class>


    </hibernate-mapping>
```

# Card.hbm.xml

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">

    <class name="Card" table="t_card">
        <id name="id" column="id">
            <generator class="foreign">
                <param name="property">person</param>
            </generator>
        </id>
        <property name="cardNum"></property>
        <one-to-one name="person" class="Person"></one-to-one>
    </class>

</hibernate-mapping>
```

# 级联删除

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">

  <class name="Person" table="t_person">
        <id name="id">
            <generator class="native"></generator>
        </id>
            <property name="userName"></property>
            <one-to-one name="card" class="Card" cascade="delete“/>
  </class>


</hibernate-mapping>
```

# cascade属性

凯盛软件

save-update：在执行保存和修改是进行级联操作

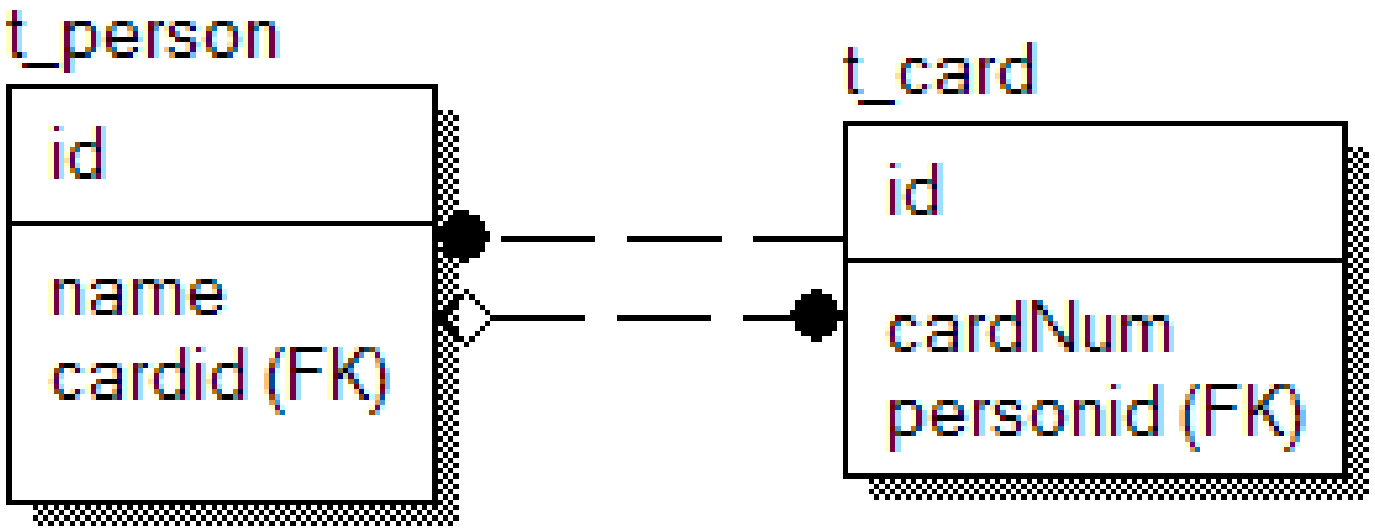
delete：在执行删除时进行级联操作

all：在所有情况下进行级联操作

none：不进行级联操作（默认）

# 一对一(唯一外键关联)

凯盛软件



# Person.hbm.xml

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">

  <class name="Person" table="t_person">
        <id name="id">
            <generator class="native"></generator>
        </id>
        <property name="userName"></property>
            <many-to-one name="card" column="cardid" class="Card"
                        cascade="delete" unique="true"/>
  </class>


</hibernate-mapping>
```

# Card.hbm.xml

凯盛软件

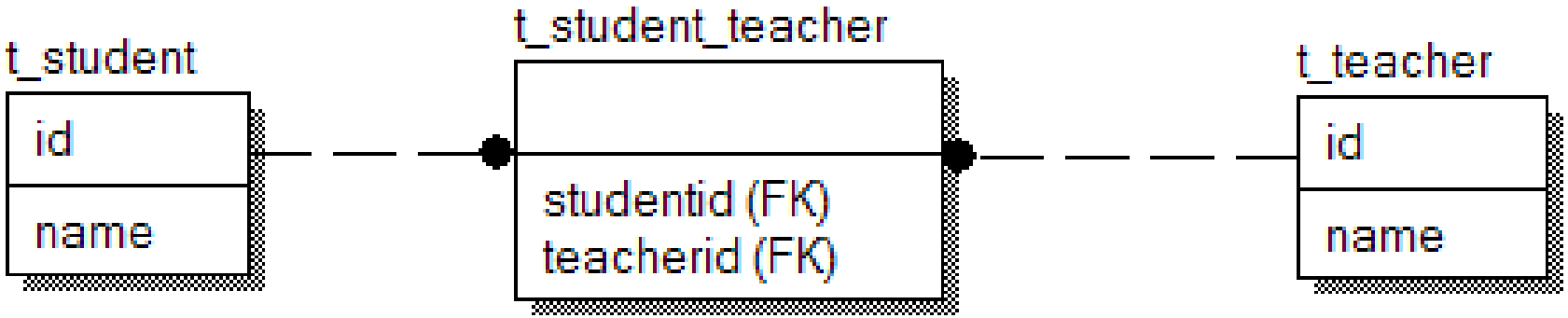
```
<hibernate-mapping package="com.kaishengit.pojo">

    <class name="Card" table="t_card">
        <id name="id" column="id">
                <generator class="native"></generator>
        </id>
        <property name="cardNum"></property>
        <many-to-one name="person" class="Person" column="personid"
                unique="true“/>
    </class>

</hibernate-mapping>
```

# 多对多映射

凯盛软件


t_student
id
name
t_student_teacher
studentid (FK)
teacherid (FK)
t_teacher
id
name

# 持久化类设计

凯盛软件

```
public class Teacher {

    private int id;
    private String name;
    private Set<Student> student;

    //get set method
}
```

```
public class Student {

    private int id;
    private String name;
    private Set<Teacher> teacher;

    //get set method
}
```

# Teacher.hbm.xml

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">
    <class name="Teacher" table="t_teacher">
        <id name="id" column="id">
            <generator class="native"></generator>
        </id>
        <property name="name"></property>
        <set name="student" table="t_student_techer">
            <key column="teacherid"/>
            <many-to-many class="Student" column="studentid"/>
        </set>
    </class>
</hibernate-mapping>
```

# Student.hbm.xml

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">
    <class name="Student" table="t_student">
        <id name="id" column="id">
                <generator class="native"></generator>
        </id>
        <property name="name"></property>
        <set name="teacher" table="t_student_techer">
            <key column="studentid"/>
            <many-to-many class="Teacher" column="teacherid"/>
        </set>
    </class>
</hibernate-mapping>
```

# 延迟加载

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">
    <class name="User" table="t_user">
        <id name="id" column="id">
                <generator class="native"></generator>
        </id>
        <property name="name" column="name"></property>
        <property name="age" column="age"></property>

        <set name="addressSet" inverse="true" lazy="false">
            <key column="userid"/>
            <one-to-many class="Address"/>
        </set>
    </class>

</hibernate-mapping>
```

# 抓取策略

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">
    <class name="User" table="t_user">
        <id name="id" column="id">
                <generator class="native"></generator>
        </id>
        <property name="name" column="name"></property>
        <property name="age" column="age"></property>

        <set name="addressSet" inverse="true" fetch="join">
            <key column="userid"/>
            <one-to-many class="Address"/>
        </set>
    </class>

</hibernate-mapping>
```

# 延迟加载范围

凯盛软件

```
session.beginTransaction();

User u = (User) session.load(User.class, 2);

System.out.println(u.getName());

session.getTransaction().commit();

Set<Address> set = u.getAddressSet();
for(Address add : set){
        System.out.println(add.getAddress());
}
```

# 解决方案

凯盛软件

关闭延迟加载功能

修改抓取策略

使用Hibernate对象的initialize方法将关联对象进行预加载

```
session.beginTransaction();

User u = (User) session.load(User.class, 2);

System.out.println(u.getName());

Hibernate.initialize(u.getAddressSet());

session.getTransaction().commit();

Set<Address> set = u.getAddressSet();
for(Address add : set){
        System.out.println(add.getAddress());
}
```

# order by

凯盛软件

```
<hibernate-mapping package="com.kaishengit.pojo">
    <class name="User" table="t_user">
        <id name="id" column="id">
                <generator class="native"></generator>
        </id>
        <property name="name" column="name"></property>
        <property name="age" column="age"></property>

        <set name="addressSet" inverse="true" order-by="id desc">
            <key column="userid"/>
            <one-to-many class="Address"/>
        </set>
    </class>

</hibernate-mapping>
```

# 缓存

凯盛软件

- 一级缓存（内置缓存）
- 二级缓存（外置缓存）

# 一级缓存

凯盛软件

一级缓存在Session中实现，当Session关闭时一级缓存就失效了。

```
session.beginTransaction();

User user = (User) session.get(User.class, 2);
User user2 = (User) session.get(User.class, 2);

session.getTransaction().commit();
```

# contains方法

凯盛软件

判断对象是否存在于一级缓存中

```
session.beginTransaction();

User user = (User) session.get(User.class, 2);
System.out.println(session.contains(user));

User user2 = (User) session.get(User.class, 2);
System.out.println(session.contains(user2));

session.getTransaction().commit();
```

# clear方法和evict方法

凯盛软件

clear方法用于将所有对象从一级缓存中清除

evict方法用于将指定对象从一级缓存中清除

```
session.beginTransaction();

User user = (User) session.get(User.class, 2);

session.evict(user);

User user2 = (User) session.get(User.class, 2);

session.getTransaction().commit();
```

# 二级缓存

凯盛软件

在Hibernate中二级缓存在SessionFactory中实现，由一个SessionFactory的所有Session实例所共享。Session在查找一个对象时，会首先在自己的一级缓存中进行查找，如果没有找到，则进入二级缓存中进行查找，如果二级缓存中存在，则将对象返回，如果二级缓存中也不存在，则从数据库中获得。

Hibernate并未提供对二级缓存的产品化实现，而是为第三方缓存组件的使用提供了接口，当前Hibernate支持的第三方二级缓存的实现如下：

- EHCache
- Proxool
- OSCache
- SwarmCache
- JBossCache

# 使用ehcache

凯盛软件

导入jar包

添加ehcache.xml

设置二级缓存

# ehcache.xml

凯盛软件

```
<ehcache>
  <diskStore path="java.io.tmpdir"/>
  <defaultCache
    maxElementsInMemory="10000" →缓存中最大允许保存的对象数量
    eternal="false" →缓存中数据是否为常量
    timeToIdleSeconds="120" →缓存数据钝化时间，单位为秒
    timeToLiveSeconds="120" →缓存数据生存时间，单位为秒
    overflowToDisk="true" →内存不足时，是否启用磁盘缓存
    />
</ehcache>
```

# 开启二级缓存

凯盛软件

hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
    …
        <property name="hibernate.cache.region.factory_class">
            org.hibernate.cache.ehcache.EhCacheRegionFactory
        </property>
    …
    </session-factory>
</hibernate-configuration>
```

xxx.hbm.xml

```
<hibernate-mapping package="com.kaishengit.pojo">
    <class name="User" table="t_user">
        <cache usage="read-only"/>
        ...
    </class>
</hibernate-mapping>
```

EHCache支持以下三种同步策略：

read-only：只读。对于不会发生改变的数据，可以使用只读性缓存。
read-write：可读写缓存。用于对数据同步要求严格的情况。
nonstrict-read-write：如果程序对并发访问下的数据同步要求不是很严格，
且数据更新操作不频繁时可采用该缓存策略

# 将对象从二级缓存中清除

凯盛软件

```
Session session = HibernateUtil.
                getSessionFactory().getCurrentSession();
session.beginTransaction();
User user = (User) session.get(User.class, 2);
session.getTransaction().commit();

Cache cache = HibernateUtil.getSessionFactory().getCache();
cache.evictEntityRegion(User.class);

Session session2 = HibernateUtil.
                getSessionFactory().getCurrentSession();
session2.beginTransaction();
User u = (User) session2.get(User.class, 2);
session2.getTransaction().commit();
```

# Hibernate Annotation

凯盛软件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>

    <session-factory>
        …
        <mapping class="com.kaishengit.pojo.User"/>
    </session-factory>

</hibernate-configuration>
```

# Pojo

凯盛软件

```
@Entity
@Table(name="t_user")
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE)
public class User {

    private int id;
    private String username;
    private String password;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    public int getId() {
        return id;
    }
```

```
    public void setId(int id) {
        this.id = id;
    }

    @Column(name="username")
    public String getUsername() {
        return username;
    }
…
```

# 一对多和多对一

凯盛软件

User.java

```
@OneToMany(mappedBy="user")
public Set<Address> getAddressSet() {
        return addressSet;
}
```

Address.java

```
@ManyToOne
@JoinColumn(name="userid")
public User getUser() {
        return user;
}
```

# 多对多

凯盛软件

Student.java

```
@ManyToMany(mappedBy="students")
public Set<Teacher> getTeachers() {
        return teachers;
}
```

Teacher.java

```
@ManyToMany
@JoinTable(name="t_student_teacher",
joinColumns=@JoinColumn(name="studentid"),
inverseJoinColumns=@JoinColumn(name="teacherid"))
public Set<Student> getStudents() {
        return students;
}
```

# 一对一

凯盛软件

User.java

```
@OneToOne
@PrimaryKeyJoinColumn
public Card getCard() {
        return card;
}
```

Card.java

```
@Id
    @GeneratedValue(generator="pkGenerator")
    @GenericGenerator(name = "pkGenerator",strategy = "foreign",parameters={@Parameter(name = "property", value =
    "user")})
    public int getId() {
        return id;
    }
    @OneToOne(mappedBy="card")
    @PrimaryKeyJoinColumn
    public User getUser() {
        return user;
    }
```

# 排序

凯盛软件

```
@ManyToMany(mappedBy="students")
@OrderBy("id desc")
public Set<Teacher> getTeachers() {
        return teachers;
}
```

# 排除不进行持久化操作的属性

凯盛软件

```
@Transient
public int getAge() {
        return age;
}
```

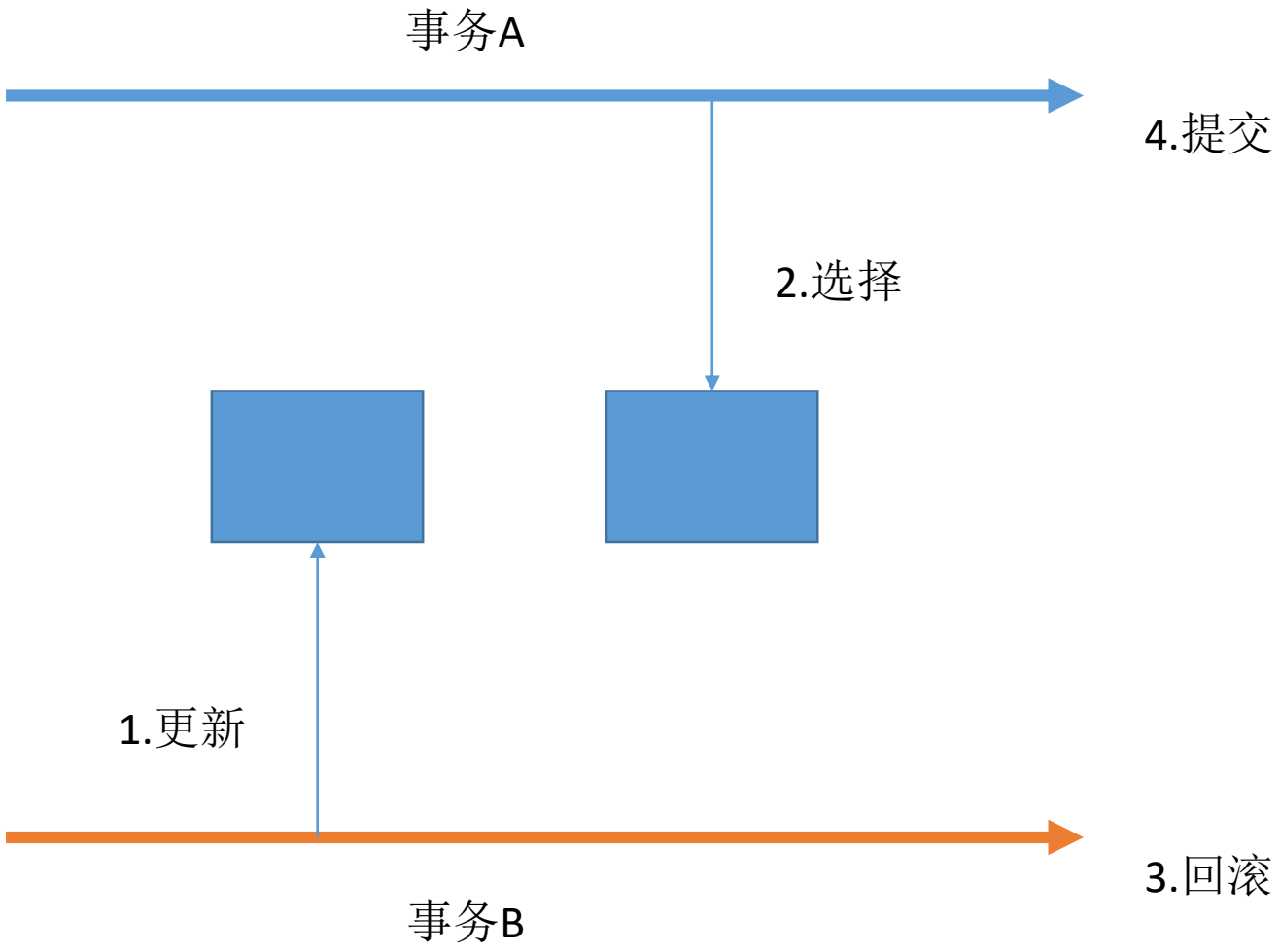
# 事务的隔离级别

凯盛软件

- 脏读
- 不可重复读
- 幻读
- 可序列化

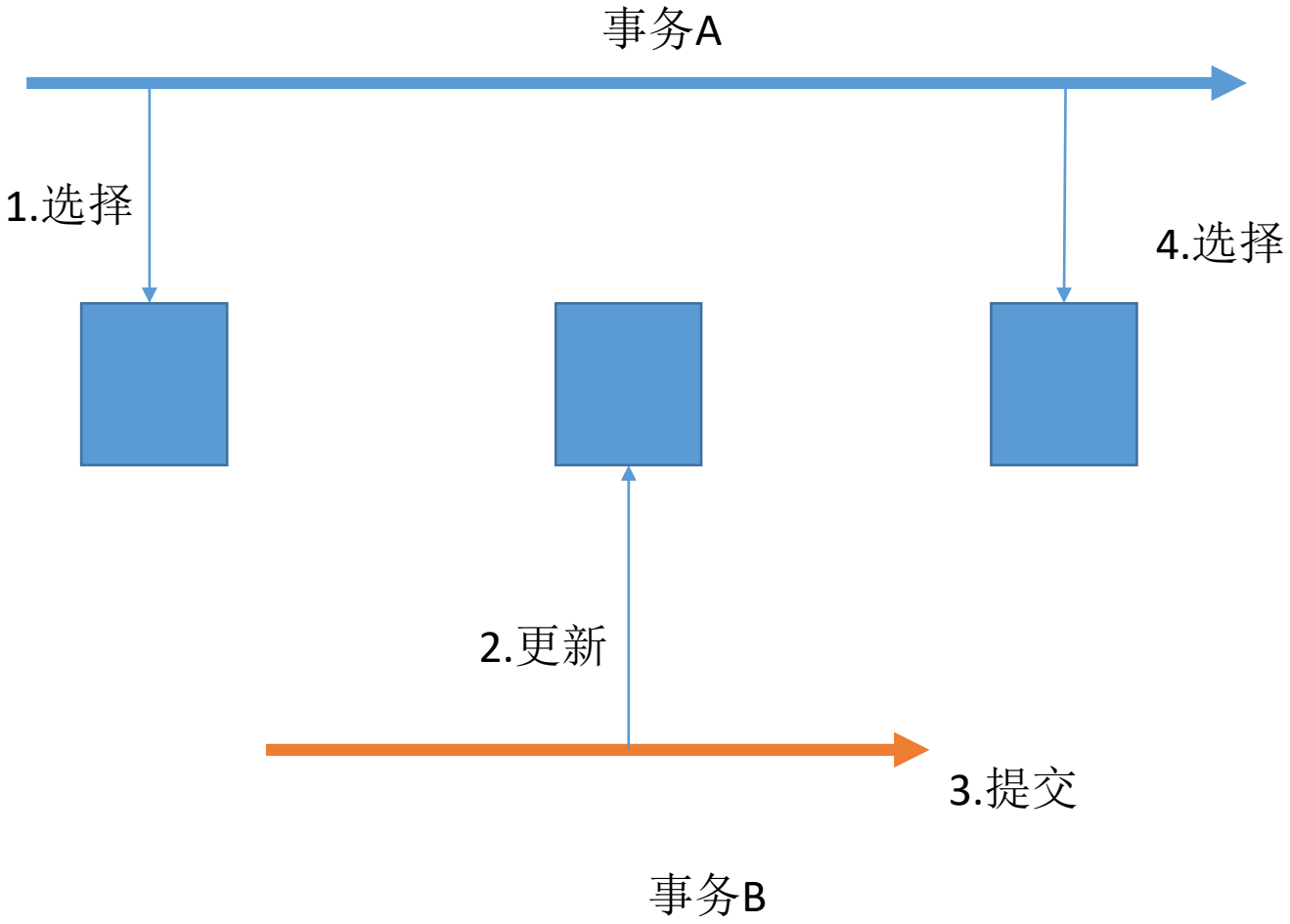
# 脏读

凯盛软件



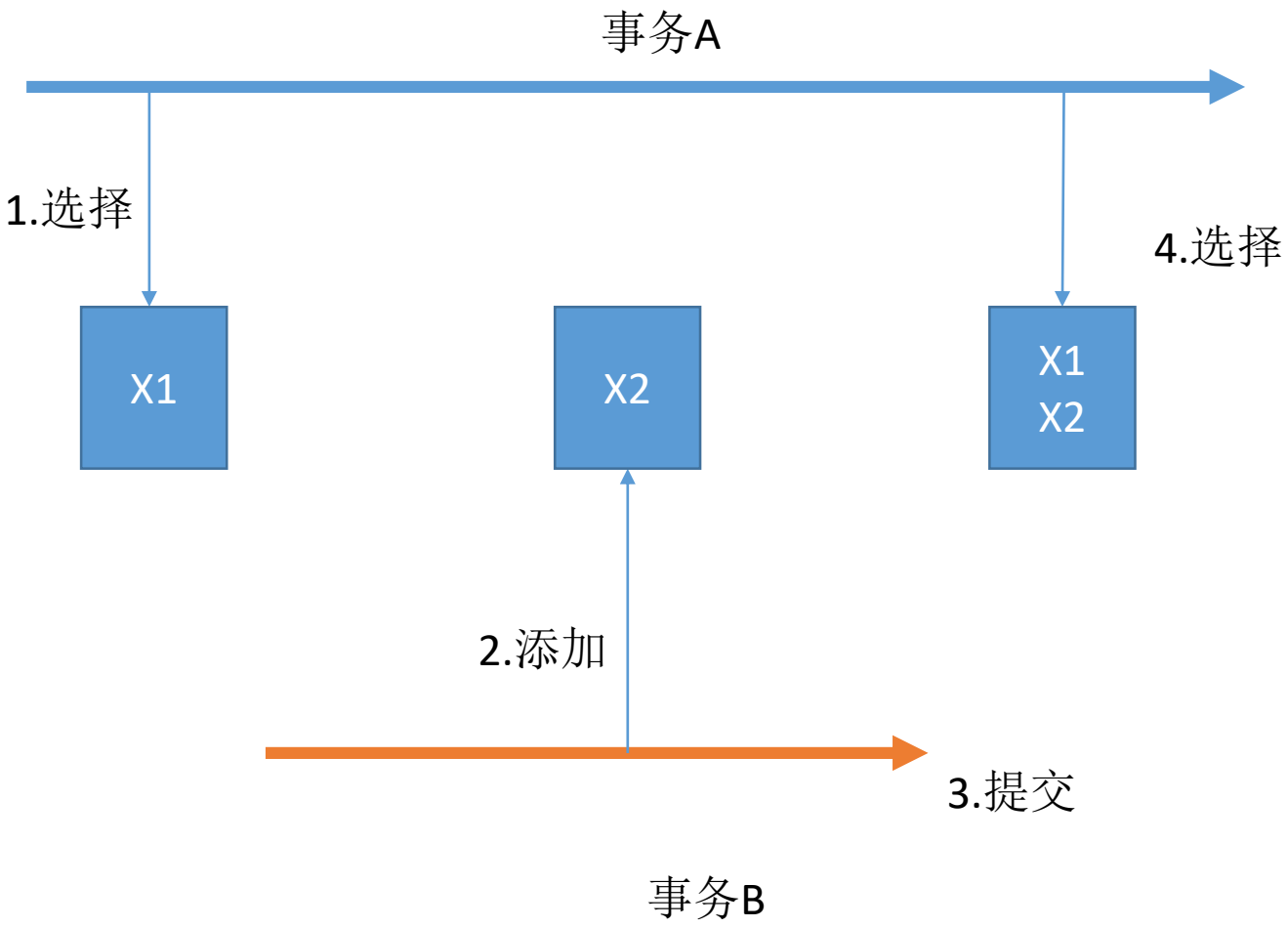
# 不可重复读

凯盛软件



# 幻读

凯盛软件



# Hibernate中的隔离级别

凯盛软件

默认设置
Hibernate.connection.isolation = 4

1：读操作未提交（Read Uncommitted）

2：读操作已提交（Read Committed）

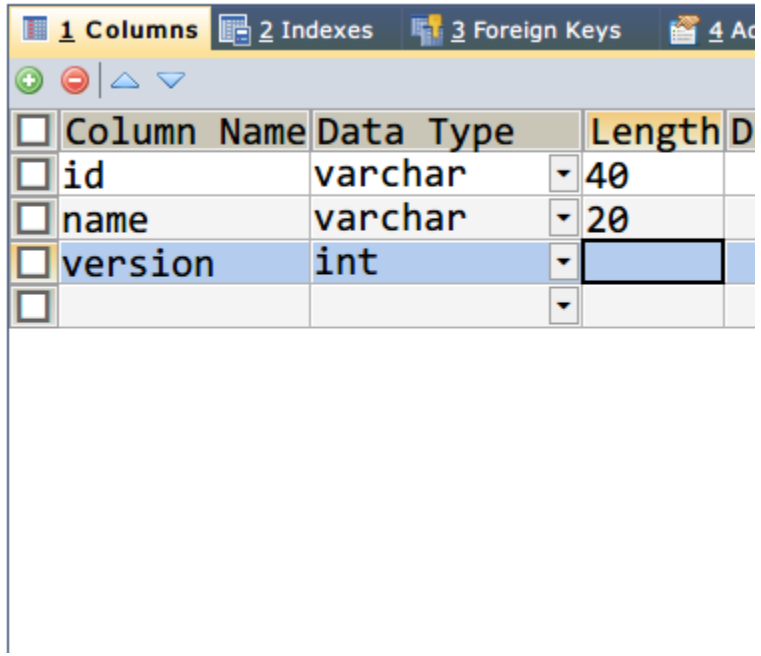
**4：可重读（Repeatable Read）**

8：可串行化（Serializable）

# 乐观锁

凯盛软件

通过version字段或timestamp字段实现




```
@Version
private Integer version;

<version name="version"/>
```

当对表数据进行添加和修改时，version字段会改变

当修改数据时，Hibernate会获取当前的version值，提交当前事务时，如果version值和之前获取的不同，那就会抛出org.hibernate.StaleObjectStateException异常。

# 悲观锁

凯盛软件

```
Session session = HibernateUtil.getSession();
session.beginTransaction();

Account account = (Account) session.get(Account.class,1,LockOptions.UPGRADE);

account.setName("aaa");

Thread thread = new Thread(new Runnable() {

    public void run() {
       Session session2 = HibernateUtil.getSession();
     session2.beginTransaction();
     Account a = (Account) session2.get(Account.class, 1);
     a.setName("bbb");
     session2.getTransaction().commit();
     System.out.println("session2 end");
    }
});

thread.start();

session.getTransaction().commit();
System.out.println("session1 end");
```

# Spring + Hibernate4

凯盛软件

```
<!-- HibernateSessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="packagesToScan" value="com.kaishengit.entity"/>
        <property name="hibernateProperties">
                <props>
                        <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                        <prop key="hibernate.show_sql">true</prop>
                </props>
        </property>
</bean>
```

```
<!-- Hibernate4事务管理器 -->
<bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory"></property>
</bean>
```

# OpenSessionInView

凯盛软件

```
<filter>
        <filter-name>openSessionInView</filter-name>
        <filter-class>
                org.springframework.orm.hibernate4.support.OpenSessionInViewFilter
        </filter-class>
</filter>
<filter-mapping>
        <filter-name>openSessionInView</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

# DAO

凯盛软件

```
@Named
public class UserDao {
        @Inject
        private SessionFactory sessionFactory;

        public void save(User user) {
                sessionFactory.getCurrentSession().save(user);
        }
}
```

# Service

凯盛软件

```
@Transactional
@Named
public class UserService {
        @Inject
        private UserDao userDao;

        public void save(User user) {
                userDao.save(user);
        }
}
```