hibernate annotation注解方式来处理映射关系 - xiaoluo501395377 - 博客园

在hibernate中，通常配置对象关系映射关系有两种，一种是基于xml的方式，另一种是基于annotation的注解方式，熟话说，萝卜青菜，可有所爱，每个人都有自己喜欢的配置方式，我在试了这两种方式以后，发现使用annotation的方式可以更简介，所以这里就简单记录下通过annotation来配置各种映射关系，在hibernate4以后已经将annotation的jar包集成进来了，如果使用hibernate3的版本就需要引入annotation的jar包。

一、单对象操作

复制代码

```
@Entity    --->    如果我们当前这个bean要设置成实体对象，就需要加上Entity这个注解@Table(name="t_user")    ---->    设置数据库的表名publicclass User{    privateint id;    private String username;    private String password;    private Date born;    private Date registerDate;
    @Column(name="register_date")    --->    Column中的name属性对应了数据库的该字段名字，里面还有其他属性，例如length，nullable等等    public Date getRegisterDate()    {        return registerDate;    }    publicvoid setRegisterDate(Date registerDate)    {        this.registerDate = registerDate;    }
    @Id    --->    定义为数据库的主键ID    (建议不要在属性上引入注解，因为属性是private的，如果引入注解会破坏其封装特性，所以建议在getter方法上加入注解)    @GeneratedValue    ---->    ID的生成策略为自动生成        publicint getId()    {    return id;    }    publicvoid setId(int id)    {        this.id = id;    }    .....}
```

复制代码

最后只需要在hibernate.cfg.xml文件里面将该实体类加进去即可：

```
        class="com.xiaoluo.bean.User"/>
```

这样我们就可以写测试类来进行我们的CRUD操作了。

二、一对多的映射(one-to-many)

这里我们定义了两个实体类，一个是ClassRoom，一个是Student，这两者是一对多的关联关系。

ClassRoom类：

复制代码

```
@Entity@Table(name="t_classroom")publicclass ClassRoom{    privateint id;    private String className;    private Set students;        public ClassRoom()    {        students = new HashSet();    }        publicvoid addStudent(Student student)    {        students.add(student);    }    @Id    @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getClassName()    {        return className;    }    publicvoid setClassName(String className)    {        this.className = className;    }
@OneToMany(mappedBy="room")    --->    OneToMany指定了一对多的关系，mappedBy="room"指定了由多的那一方来维护关联关系，mappedBy指的是多的一方对1的这一方的依赖的属性，(注意：如果没有指定由谁来维护关联关系，则系统会给我们创建一张中间表)    @LazyCollection(LazyCollectionOption.EXTRA)    --->    LazyCollection属性设置成EXTRA指定了当如果查询数据的个数时候，只会发出一条 count(*)的语句，提高性能    public Set getStudents()    {        return students;    }    publicvoid setStudents(Set students)    {        this.students = students;    }    }
```

复制代码

Student类：

复制代码

```
@Entity@Table(name="t_student")publicclass Student{    privateint id;    private String name;    privateint age;    private ClassRoom room;        @ManyToOne(fetch=FetchType.LAZY)    ---> ManyToOne指定了多对一的关系，fetch=FetchType.LAZY属性表示在多的那一方通过延迟加载的方式加载对象(默认不是延迟加载)    @JoinColumn(name="rid")    --->    通过 JoinColumn 的name属性指定了外键的名称 rid (注意：如果我们不通过JoinColum来指定外键的名称，系统会给我们声明一个名称)    public ClassRoom getRoom()    {        return room;    }    publicvoid setRoom(ClassRoom room)    {    this.room = room;    }    @Id    @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getName()    {        return name;    }    publicvoid setName(String name)    {        this.name = name;    }    publicint getAge()    {        return age;    }    publicvoid setAge(int age)    {        this.age = age;    }    }
```

复制代码

三、一对一映射(One-to-One)

一对一关系这里定义了一个Person对象以及一个IDCard对象

Person类：

复制代码

```
@Entity@Table(name="t_person")publicclass Person{    privateint id;    private String name;    private IDCard card;    @OneToOne(mappedBy="person")    --->    指定了OneToOne的关联关系，mappedBy同样指定由对方来进行维护关联关系    public IDCard getCard()    {        return card;    }    publicvoid setCard(IDCard card)    {        this.card = card;    }    @Id    @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getName()    {        return name;    }    publicvoid setName(String name)    {        this.name = name;    }    }
```

复制代码

IDCard类：

复制代码

```
@Entity@Table(name="t_id_card")publicclass IDCard{    privateint id;    private String no;    private Person person;    @Id    @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getNo()    {        return no;    }    publicvoid setNo(String no)    {        this.no = no;    }    @OneToOne    --->    OnetoOne指定了一对一的关联关系，一对一中随便指定一方来维护映射关系，这里选择IDCard来进行维护    @JoinColumn(name="pid")    --->    指定外键的名字 pid    public Person getPerson()    {        return person;    }    publicvoid setPerson(Person person)    {        this.person = person;    }}
```

复制代码

注意:在判断到底是谁维护关联关系时，可以通过查看外键，哪个实体类定义了外键，哪个类就负责维护关联关系。

四、Many-to-Many映射(多对多映射关系)

多对多这里通常有两种处理方式，一种是通过建立一张中间表，然后由任一一个多的一方来维护关联关系，另一种就是将多对多拆分成两个一对多的关联关系

1.通过中间表由任一一个多的一方来维护关联关系

Teacher类：

复制代码

```
@Entity@Table(name="t_teacher")publicclass Teacher{    privateint id;    private String name;    private Set courses;    public Teacher()    {        courses = new HashSet();    }    publicvoid addCourse(Course course)    {        courses.add(course);    }        @Id    @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getName()    {        return name;    }    publicvoid setName(String name)    {        this.name = name;    }    @ManyToMany(mappedBy="teachers")    --->    表示由Course那一方来进行维护    public Set getCourses()    {        return courses;    }    publicvoid setCourses(Set courses)    {        this.courses = courses;    }    }
```

复制代码

Course类：

复制代码

```
@Entity@Table(name="t_course")publicclass Course{    privateint id;    private String name;    private Set teachers;    public Course()    {        teachers = new HashSet();    }    publicvoid addTeacher(Teacher teacher)    {        teachers.add(teacher);    }    @ManyToMany        ---> ManyToMany指定多对多的关联关系    @JoinTable(name="t_teacher_course", joinColumns={ @JoinColumn(name="cid")},        inverseJoinColumns={ @JoinColumn(name="tid") })    --->    因为多对多之间会通过一张中间表来维护两表直接的关系，所以通过 JoinTable这个注解来声明，name就是指定了中间表的名字，JoinColumns是一个 @JoinColumn类型的数组，表示的是我这方在对方中的外键名称，我方是Course，所以在对方外键的名称就是 rid，inverseJoinColumns也是一个 @JoinColumn类型的数组，表示的是对方在我这放中的外键名称，对方是Teacher，所以在我方外键的名称就是 tid    public Set getTeachers()    {        return teachers;    }    publicvoid setTeachers(Set teachers)    {        this.teachers = teachers;    }    @Id    @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getName()    {        return name;    }    publicvoid setName(String name)    {        this.name = name;    }}
```

复制代码

2.将Many-to-Many拆分成两个One-to-Many的映射(Admin、Role、AdminRole)

Admin类：

复制代码

```
@Entity@Table(name="t_admin")publicclass Admin{    privateint id;    private String name;    private Set ars;    public Admin()    {        ars = new HashSet();    }    publicvoid add(AdminRole ar)    {        ars.add(ar);    }    @Id @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getName()    {        return name;    }    publicvoid setName(String name)    {        this.name = name;    }    @OneToMany(mappedBy="admin")    --->    OneToMany关联到了AdminRole这个类，由AdminRole这个类来维护多对一的关系，mappedBy="admin"    @LazyCollection(LazyCollectionOption.EXTRA)        public Set getArs()    {        return ars;    }    publicvoid setArs(Set ars)    {        this.ars = ars;    }}
```

复制代码

Role类：

复制代码

```
@Entity@Table(name="t_role")publicclass Role{    privateint id;    private String name;    private Set ars;    public Role()    {        ars = new HashSet();    }    publicvoid add(AdminRole ar)    {        ars.add(ar);    }    @Id @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getName()    {        return name;    }    publicvoid setName(String name)    {        this.name = name;    }    @OneToMany(mappedBy="role")    --->    OneToMany指定了由AdminRole这个类来维护多对一的关联关系，mappedBy="role"    @LazyCollection(LazyCollectionOption.EXTRA)    public Set getArs()    {        return ars;    }    publicvoid setArs(Set ars)    {        this.ars = ars;    }}
```

复制代码

AdminRole类：

复制代码

```
@Entity@Table(name="t_admin_role")publicclass AdminRole{    privateint id;    private String name;    private Admin admin;    private Role role;    @Id    @GeneratedValue    publicint getId()    {        return id;    }    publicvoid setId(int id)    {        this.id = id;    }    public String getName()    {        return name;    }    publicvoid setName(String name)    {        this.name = name;    }    @ManyToOne    --->    ManyToOne关联到Admin @JoinColumn(name="aid")        public Admin getAdmin()    {        return admin;    }    publicvoid setAdmin(Admin admin)    {        this.admin = admin;    }    @ManyToOne    --->        @JoinColumn(name="rid")    public Role getRole()    {        return role;    }    publicvoid setRole(Role role)    {        this.role = role;    }}
```

复制代码

小技巧:通过hibernate来进行插入操作的时候，不管是一对多、一对一还是多对多，都只需要记住一点，在哪个实体类声明了外键，就由哪个类来维护关系，在保存数据时，总是先保存的是没有维护关联关系的那一方的数据，后保存维护了关联关系的那一方的数据，如：

复制代码

```
Person p = new Person();            p.setName("xiaoluo");            session.save(p);
IDCard card = new IDCard();            card.setNo("1111111111");            card.setPerson(p);
session.save(card);
```

复制代码

以上就是对hibernate annotation注解方式来配置映射关系的一些总结。