JPA注解的使用

基本注解

基本注解 @Entity、@Table、@Id、@GeneratedValue、@Basic、@Column、@Transient、@Lob、@Temporal

先看一个 Blog 的案例其中实体的配置如下：

```
@Entity
@Table(name = "user_blog", schema = "test")
public class UserBlogEntity {
   @Id
   @Column(name = "id", nullable = false)
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Integer id;
   @Column(name = "title", nullable = true, length = 200)
   private String title;
   @Basic
   @Column(name = "create_user_id", nullable = true)
   private Integer createUserId;
   @Basic
   @Column(name = "blog_content", nullable = true, length = -1)
   @Lob
   private String blogContent;
   @Basic(fetch = FetchType.LAZY)
   @Column(name = "image", nullable = true)
   @Lob
   private byte[] image;
   @Basic
   @Column(name = "create_time", nullable = true)
   @Temporal(TemporalType.TIMESTAMP)
   private Date createTime;
   @Basic
   @Column(name = "create_date", nullable = true)
   @Temporal(TemporalType.DATE)
   private Date createDate;
   @Transient
   private String transientSimple;
}
```

下面对上面类中用到的注解来一一解释一下:

@Entity

@Entity 用于定义对象将会成为被 JPA 管理的实体，将字段映射到指定的数据库表中，源码如下:

```
public @interface Entity {
   //可选，默认是次实体类的名字，全局唯一。
   String name() default "";
}
```

@Table

@Table 用于指定数据库的表名

```
public @interface Table {
   //表的名字，可选。如果不填写，系统认为好实体的名字一样为表名。
   String name() default "";
   //此表的catalog，可选
   String catalog() default "";
   //此表所在schema，可选
   String schema() default "";
   //唯一性约束，只有创建表的时候有用，默认不需要。
   UniqueConstraint[] uniqueConstraints() default { };
   //索引，只有创建表的时候使用，默认不需要。
   Index[] indexes() default {};
}
```

@Id

@Id 定义属性为数据库的主键，一个实体里面必须有一个，并且必须和 @GeneratedValue 配合使用和成对出现

@GeneratedValue

@GeneratedValue 主键生成策略

```
public @interface GeneratedValue {
    //Id的生成策略
    GenerationType strategy() default AUTO;
    //通过Sequences生成Id,常见的是Orcale数据库ID生成规则，这个时候需要配合@SequenceGenerator使用
    String generator() default "";
}
```

GenerationType 一共有以下四个值：

```
public enum GenerationType {
    //通过表产生主键，框架借由表模拟序列产生主键，使用该策略可以使应用更易于数据库移植。
    TABLE,
    //通过序列产生主键，通过 @SequenceGenerator 注解指定序列名， MySql 不支持这种方式；
    SEQUENCE,
    //采用数据库ID自增长， 一般用于mysql数据库
    IDENTITY,
    //JPA 自动选择合适的策略，是默认选项；
    AUTO
}
```

@IdClass

@IdClass 利用外部类的联合主键，源码：

```
public @interface IdClass {
//联合主键的类
    Class value();
}
```

作为复合主键类，要满足以下几点要求。

- 必须实现 Serializable 接口。
- 必须有默认的 public 无参数的构造方法。
- 必须覆盖 equals 和 hashCode 方法。equals 方法用于判断两个对象是否相同，EntityManger 通过 find 方法来查找 Entity 时，是根据 equals 的返回值来判断的。本例中，只有对象的 name 和 email 值完全相同时或同一个对象时则返回 true，否则返回 false。hashCode 方法返回当前对象的哈希码，生成 hashCode 相同的概率越小越好，算法可以进行优化。

@IdClass的用法

假设 UserBlog 的联合主键是 createUserId 和 title，新增一个 UserBlogKey 的类。

UserBlogKey.class

```
import java.io.Serializable;
public class UserBlogKey implements Serializable {
   private String title;
   private Integer createUserId;
   public UserBlogKey() {
   }
   public UserBlogKey(String title, Integer createUserId) {
      this.title = title;
      this.createUserId = createUserId;
   }
    //get set 方法我们略过
}
```

UserBlogEntity.java 稍加改动，实体类上需要加 @IdClass 注解和两个主键上都得加 @Id 注解，如下。

```
@Entity
@Table(name = "user_blog", schema = "test")
@IdClass(value = UserBlogKey.class)
public class UserBlogEntity {
   @Column(name = "id", nullable = false)
   private Integer id;
   @Id
   @Column(name = "title", nullable = true, length = 200)
   private String title;
   @Id
   @Column(name = "create_user_id", nullable = true)
   private Integer createUserId;
    //不变的部分我们省略
}
```

UserBlogRepository 我们做的改动：

```
public interface UserBlogRepository extends JpaRepository<UserBlogEntity,UserBlogKey>{
}
```

使用的时候：

```
@RequestMapping(path = "/blog/{title}/{createUserId}")
@ResponseBody
public Optional<UserBlogEntity> showBlogs(@PathVariable(value = "createUserId") Integer
createUserId,@PathVariable("title") String title) {
   return userBlogRepository.findById(new UserBlogKey(title,createUserId));
}
```

@Basic

@Basic 表示属性是到数据库表的字段的映射。如果实体的字段上没有任何注解，默认即为 @Basic。

```
public @interface Basic {
    //可选，EAGER（默认）：立即加载；LAZY：延迟加载。（LAZY主要应用在大字段上面）
    FetchType fetch() default EAGER;
    //可选。这个字段是否可以为null，默认是true。
    boolean optional() default true;
}
```

@Transient

@Transient 表示该属性并非一个到数据库表的字段的映射，表示非持久化属性。JPA 映射数据库的时候忽略它，与 @Basic 相反的作用

@Column

@Column 定义该属性对应数据库中的列名

```
public @interface Column {
    //数据库中的表的列名；可选，如果不填写认为字段名和实体属性名一样。
    String name() default "";
    //是否唯一。默认flase，可选。
    boolean unique() default false;
    //数据字段是否允许空。可选，默认true。
    boolean nullable() default true;
    //执行insert操作的时候是否包含此字段，默认，true，可选。
    boolean insertable() default true;
    //执行update的时候是否包含此字段，默认，true，可选。
    boolean updatable() default true;
    //表示该字段在数据库中的实际类型。
    String columnDefinition() default "";
   //数据库字段的长度，可选，默认255
    int length() default 255;
}
```

@Temporal

@Temporal 用来设置 Date 类型的属性映射到对应精度的字段。

- @Temporal(TemporalType.DATE)映射为日期 // date （只有日期）
- @Temporal(TemporalType.TIME)映射为日期 // time （是有时间）
- @Temporal(TemporalType.TIMESTAMP)映射为日期 // date time （日期+时间）

@Lob

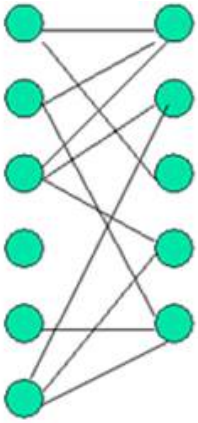
@Lob 将属性映射成数据库支持的大对象类型，支持以下两种数据库类型的字段

- Clob（Character Large Ojects）类型是长字符串类型，java.sql.Clob、Character[]、char[] 和 String 将被映射为 Clob 类型。
- Blob（Binary Large Objects）类型是字节类型，java.sql.Blob、Byte[]、byte[]和实现了 Serializable 接口的类型将被映射为 Blob 类型。
- 由于 Clob，Blob 占用内存空间较大一般配合 @Basic(fetch=FetchType.LAZY) 将其设置为延迟加载。

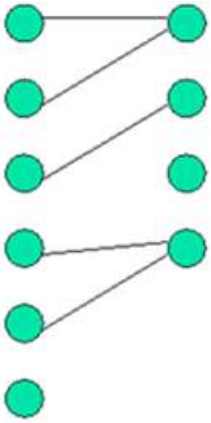
多表设计

表之间关系划分

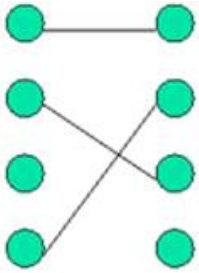
数据库中多表之间存在着三种关系，如图所示：



多对多
M：N



一对多
1：N



一对一
1：1

> 从图可以看出，系统设计的三种实体关系分别为：多对多、一对多和一对一关系。注意：一对多关系可以看为两种： 即一对多，多对一。所以说四种更精确。

JPA表关系分析步骤

> 在实际开发中，我们数据库的表难免会有相互的关联关系，在操作表的时候就有可能会涉及到多张表的操作。而在这种实现了ORM思想的框架中（如JPA），可以让我们通过操作实体类就实现对数据库表的操作。所以今天我们的学习重点是：掌握配置实体之间的关联关系。
>
> **第一步：首先确定两张表之间的关系。**
>
> 如果关系确定错了，后面做的所有操作就都不可能正确。
>
> **第二步：在数据库中实现两张表的关系**
>
> **第三步：在实体类中描述出两个实体的关系**
>
> **第四步：配置出实体类和数据库表的关系映射（重点）**

关联关系注解

> @OneToOne、@JoinColumn、@ManyToOne、@ManyToMany、@JoinTable、@OrderBy

@JoinColumn 定义外键关联的字段名称

源码语法

```
public @interface JoinColumn {
    //目标表的字段名,必填
    String name() default "";
    //本实体的字段名，非必填，默认是本表ID
    String referencedColumnName() default "";
    //外键字段是否唯一
    boolean unique() default false;
    //外键字段是否允许为空
    boolean nullable() default true;
    //是否跟随一起新增
    boolean insertable() default true;
    //是否跟随一起更新
    boolean updatable() default true;
}
```

用法

> 用法：@JoinColumn 主要配合 @OneToOne、@ManyToOne、@OneToMany 一起使用，单独使用没有意义。
>
> @JoinColumn 定义多个字段的关联关系。

@OneToOne 一对一关联关系

源码语法

```
public @interface OneToOne {
    //关系目标实体，非必填，默认该字段的类型。
    Class targetEntity() default void.class;
    //cascade 级联操作策略
    /*
          1. CascadeType.PERSIST 级联新建
          2. CascadeType.REMOVE 级联删除
          3. CascadeType.REFRESH 级联刷新
          4. CascadeType.MERGE 级联更新
          5. CascadeType.ALL 四项全选
          6. 默认，关系表不会产生任何影响
     */
    CascadeType[] cascade() default {};
    //数据获取方式EAGER(立即加载)/LAZY(延迟加载)
    FetchType fetch() default EAGER;
    //是否允许为空
    boolean optional() default true;
    //关联关系被谁维护的。 非必填，一般不需要特别指定。
//注意：只有关系维护方才能操作两者的关系。被维护方即使设置了维护方属性进行存储也不会更新外键关联。1）mappedBy不能与@JoinColumn或者@JoinTable同时使用。2）mappedBy的值是指另一方的实体里面属性的字段，而不是数据库字段，也不是实体的对象的名字。既是另一方配置了@JoinColumn或者@JoinTable注解的属性的字段名称。
    String mappedBy() default "";
    //是否级联删除。和CascadeType.REMOVE的效果一样。两种配置了一个就会自动级联删除
    boolean orphanRemoval() default false;
}
```

用法

> 用法 @OneToOne 需要配合 @JoinColumn 一起使用。注意：可以双向关联，也可以只配置一方，看实际需求

案例：假设一个学生对应一个班级，添加学生的同时添加班级，Student类的内容如下：

```
@OneToOne(cascade = CascadeType.PERSIST)
//关联的外键字段
@JoinColumn(name = "grade_id")
private Grade grade;
```

> 注意：`grade_id` 指的是t_student表中的字段，cascade属性设置级联操作

如果需要双向关联，Grade类的内容如下：

```
@OneToOne(mappedBy = "grade")
private Student student;
```

一对一操作

> 需求：学生与班级的一对一关系
>
> 学生：一方
>
> 班级：一方

Student实体类

```java
import javax.persistence.*;

@Entity
@Table(name = "t_student")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String studentname;

    //一对一关联
    @OneToOne(cascade = CascadeType.PERSIST)
    //关联的外键字段
    @JoinColumn(name = "grade_id")
    private Grade grade;

    public Grade getGrade() {
        return grade;
    }

    public void setGrade(Grade grade) {
        this.grade = grade;
    }

    private String sex;

    public Integer getId() {
        return id;
    }

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

    public String getStudentname() {
        return studentname;
    }

    public void setStudentname(String studentname) {
        this.studentname = studentname;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", studentname='" + studentname + '\'' +
                ", sex='" + sex + '\'' +
                '}';
    }

}
```

Grade实体类

```java
import javax.persistence.*;

@Entity
@Table(name = "t_grade")
public class Grade {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String gradename;

    //一对一
    @OneToOne(mappedBy = "grade")
    private Student student;

    public Student getStudent() {
        return student;
    }

    public void setStudent(Student student) {
        this.student = student;
    }

    public Integer getId() {
        return id;
    }

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

    public String getGradename() {
        return gradename;
    }

    public void setGradename(String gradename) {
        this.gradename = gradename;
    }

    @Override
    public String toString() {
        return "Grade{" +
                "id=" + id +
                ", gradename='" + gradename + '\'' +
                '}';
    }
}
```

StudentRepository接口

```
public interface StudentRepository extends JpaRepository<Student,Integer> {
}
```

测试类

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class OneToOneTest {

    @Resource
    private StudentRepository studentRepository;

    /**
     * 一对一添加：添加学生同时添加年级
     */
    @Test
    public void testAdd(){
        //创建年级对象
        Grade grade = new Grade();
        grade.setGradename("一年级");
        //创建学生对象
        Student student = new Student();
        student.setStudentname("张浩");
        student.setSex("男");
```

```
        //设置关联关系
        student.setGrade(grade);
        //保存
        studentRepository.save(student);
    }

    /**
     * 根据学生 ID 查询学生，同时查询年级
     */
    @Test
    public void testSearch(){
        Optional<Student> student = studentRepository.findById(1);
        System.out.println("学生信息："+student);
        System.out.println("年级信息："+student.get().getGrade());
    }

}
```

@OneToMany 一对多 & @ManyToOne 多对一

@OneToMany源码语法

```
public @interface OneToMany {
    Class targetEntity() default void.class;
 //cascade 级联操作策略：(CascadeType.PERSIST、CascadeType.REMOVE、CascadeType.REFRESH、
CascadeType.MERGE、CascadeType.ALL)
如果不填，默认关系表不会产生任何影响。
    CascadeType[] cascade() default {};
//数据获取方式EAGER(立即加载)/LAZY(延迟加载)
    FetchType fetch() default LAZY;
    //关系被谁维护，单项的。注意：只有关系维护方才能操作两者的关系。
    String mappedBy() default "";
//是否级联删除。和CascadeType.REMOVE的效果一样。两种配置了一个就会自动级联删除
    boolean orphanRemoval() default false;
}
public @interface ManyToOne {
    Class targetEntity() default void.class;
    CascadeType[] cascade() default {};
    FetchType fetch() default EAGER;
    boolean optional() default true;
}
```

> @ManyToOne 与 OneToMany 的源码稍有区别仔细体会

一对多的关联关系

> 需求：从角色到用户的一对多的关联关系 角色：一方 用户：多方

创建Users实体类

> 1. 多个用户拥有同一个角色（多对一）
> 2. toString()方法不添加Roles属性
> 3. 添加无参构造方法及带参(用户名，角色类型)构造方法

```
package com.kazu.springdatajpa02.entity;

import javax.persistence.*;

@Entity
@Table(name = "t_users")
public class Users {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String username;

    //多个用户拥有同一个角色
    @ManyToOne
    @JoinColumn(name = "roles_id")
    private Roles roles;

    public Integer getId() {
        return id;
    }

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

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public Roles getRoles() {
        return roles;
    }

    public void setRoles(Roles roles) {
        this.roles = roles;
    }

    @Override
    public String toString() {
        return "Users{" +
                "id=" + id +
                ", username='" + username + '\'' +
                '}';
    }

    public Users(String username,Roles roles) {
        this.username = username;
        this.roles = roles;
    }

    public Users() {
    }
}
```

创建Roles实体类

> 1. 一个角色下有多个用户（一对多）
> 2. toString()方法不添加users属性
> 3. 添加无参构造方法及带参(角色名称)构造方法

```
package com.kazu.springdatajpa02.entity;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "t_roles")
public class Roles {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String rolename;
```

```
    //一个角色被多个用户拥有
    //fetch = FetchType.EAGER:立即加载
    @OneToMany(mappedBy = "roles",cascade = CascadeType.ALL,fetch = FetchType.EAGER)
    private Set<Users> users = new HashSet<Users>();

    public Integer getId() {
        return id;
    }

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

    public String getRolename() {
        return rolename;
    }

    public void setRolename(String rolename) {
        this.rolename = rolename;
    }

    public Set<Users> getUsers() {
        return users;
    }

    public void setUsers(Set<Users> users) {
        this.users = users;
    }

    @Override
    public String toString() {
        return "Roles{" +
                "id=" + id +
                ", rolename='" + rolename + '\'' +
                '}';
    }

    public Roles() {
    }

    public Roles(String rolename) {
        this.rolename = rolename;
    }
}
```

创建RolesRepository接口

```
public interface RolesRepository extends JpaRepository<Roles,Integer> {
}
```

测试类

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class OneToManyTest {

   @Resource
   private RolesRepository rolesRepository;


    /**
     * 级联添加：添加角色同时添加用户
     */
   @Test
    public void test1(){
       //创建角色对象
       Roles roles = new Roles("经理");
```

```
        //创建两个用户
        Users users1 = new Users("王五",roles);
        Users users2 = new Users("赵六",roles);
        //设置关联关系
        roles.getUsers().add(users1);
        roles.getUsers().add(users2);
        //级联保存用户
        rolesRepository.save(roles);
    }

     /**
      * 级联查询：查询角色同时查询角色下的用户列表（关闭延迟加载）
      */
    @Test
    public void test2(){
        Optional<Roles> roles = rolesRepository.findById(1);
        System.out.println("角色名称："+roles.get().getRolename());
        for (Users users :roles.get().getUsers()){
            System.out.println(users);
        }
    }
}
```

@OrderBy 关联查询的时候的排序

> 一般和 @OneToMany 一起使用

源码语法

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface OrderBy {
    /**
     * 要排序的字段，格式如下：
     *    orderby_list::= orderby_item [,orderby_item]*
     *    orderby_item::= [property_or_field_name] [ASC | DESC]
     * 字段可以是实体属性，也可以数据字段，默认ASC。
     */
     String value() default "";
}
```

用法

```
@Entity
@Table(name="user")
public class User implements Serializable{
@OneToMany(cascade=CascadeType.ALL,fetch=FetchType.LAZY,mappedBy="user")
@OrderBy("role_name DESC")
    private Set<role> setRole;
}
```

@JoinTable 关联关系表

> @JoinTable 是指如果对象与对象之间有个关联关系表的时候，就会用到这个，一般和 @ManyToMany 一起使用。

源码语法

```
public @interface JoinTable {
    //中间关联关系表明
    String name() default "";
    //表的catalog
    String catalog() default "";
    //表的schema
    String schema() default "";
    //主链接表的字段
    JoinColumn[] joinColumns() default {};
    //被联机的表外键字段
    JoinColumn[] inverseJoinColumns() default {};
}
```

用法

> 假设 Blog 和 Tag 是多对多的关系，有个关联关系表 `blog_tag_relation` ，表中有两个属性 `blog_id` 和 `tag_id` ，那么 Blog 实体里面的写法如下：

```
@Entity
public class Blog{
    @ManyToMany
    @JoinTable(
        name="blog_tag_relation",
        joinColumns=@JoinColumn(name="blog_id",referencedColumnName="id"),
        inverseJoinColumn=@JoinColumn(name="tag_id",referencedColumnName="id")
        private List<Tag> tags = new ArrayList<Tag>();
    )
}
```

@ManyToMany 多对多

源码语法

```
public @interface ManyToMany {
    Class targetEntity() default void.class;
    CascadeType[] cascade() default {};
    FetchType fetch() default LAZY;
    String mappedBy() default "";
}
```

> @ManyToMany 表示多对多，和 @OneToOne、@ManyToOne 一样也有单向双向之分，单项双向和注解没有关系，只看实体类之间是否相互引用。 主要注意的是当用到 @ManyToMany 的时候一定是三张表，不要想着偷懒，否则会发现有很多麻烦.

多对多的关联关系

> 需求：一个项目由多个员工负责，一个员工参与多个项目（多对多关系）--给项目分配员工
>
> 员工：多方
>
> 项目：多方

创建Project实体类

> 1. 一个项目由有多个员工负责（一对多）
> 2. toString()方法不添加Employee属性
> 3. 添加无参构造方法及带参(项目名称)构造方法

```
@Entity
@Table(name = "t_project")
```

```java
public class Project {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer projectid;
    private String projectname;

    /**
     * 配置项目到员工的多对多关系
     *      配置多对多的映射关系
     *          1.声明表关系的配置
     *              @ManyToMany(targetEntity = Employee.class)  //多对多
     *                  targetEntity：代表对方的实体类字节码
     *          2.配置中间表（包含两个外键）
     *                @JoinTable
     *                  name ： 中间表的名称
     *                  joinColumns：配置当前对象在中间表的外键
     *                      @JoinColumn的数组
     *                          name：外键名
     *                          referencedColumnName：参照的主表的主键名
     *                  inverseJoinColumns：配置对方对象在中间表的外键
     */
    @ManyToMany(targetEntity =Employee.class ,cascade = CascadeType.ALL)
    //第三张表（外键关系表、中间表）
    //name属性：第三张表的表名称
    @JoinTable(name = "t_employee_project",
            //joinColumns：当前对象在中间表中的外键
            joinColumns = @JoinColumn(name = "project_id",referencedColumnName = "projectid"),
            //inverseJoinColumns:对方对象在中间表的外键
            inverseJoinColumns = @JoinColumn(name = "employee_id",referencedColumnName = "empid"))
    private Set<Employee> employees = new HashSet<Employee>();

    public Integer getProjectid() {
        return projectid;
    }

    public void setProjectid(Integer projectid) {
        this.projectid = projectid;
    }

    public String getProjectname() {
        return projectname;
    }

    public void setProjectname(String projectname) {
        this.projectname = projectname;
    }

    public Set<Employee> getEmployees() {
        return employees;
    }

    public void setEmployees(Set<Employee> employees) {
        this.employees = employees;
    }

    public Project() {
    }

    public Project(String projectname) {
        this.projectname = projectname;
    }

    @Override
    public String toString() {
        return "Project{" +
                "projectid=" + projectid +
                ", projectname='" + projectname + '\'' +
                '}';
    }
}
```

创建Employee实体类

> 1. 一个员工负责多个项目（一对多）
> 2. toString()方法不添加Project属性
> 3. 添加无参构造方法及带参(员工名称)构造方法

```
@Entity
@Table(name = "t_employee")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer empid;
    private String empname;
    //多对多放弃维护权：被动的一方放弃
    @ManyToMany(mappedBy = "employees")
    private Set<Project> projects = new HashSet<Project>();

    public Integer getEmpid() {
        return empid;
    }

    public void setEmpid(Integer empid) {
        this.empid = empid;
    }

    public String getEmpname() {
        return empname;
    }

    public void setEmpname(String empname) {
        this.empname = empname;
    }

    public Set<Project> getProjects() {
        return projects;
    }

    public void setProjects(Set<Project> projects) {
        this.projects = projects;
    }

    public Employee() {
    }

    public Employee(String empname) {
        this.empname = empname;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "empid=" + empid +
                ", empname='" + empname + '\'' +
                '}';
    }
}
```

创建ProjectRepository接口

```
public interface ProjectRepository extends JpaRepository<Project,Integer> {
}
```

测试多对多

> 测试级联添加：给项目分配员工

思路：

- 创建2个项目（超市管理系统、酒店管理系统），3个员工（张三、李四、王五）
- 张三、李四、王五负责超市管理系统
- 张三、李四同时负责酒店管理系统

```
/**
 * 级联添加：添加角色同时添加用户
 */
@Test
//以下两个注解必须提供
@Transactional//开启事务
@Rollback(false)//取消回滚
public void testAdd() {
    //创建两个项目对象
    Project project1 = new Project("超市管理系统");
    Project project2 = new Project("酒店管理系统");
    //创建三个员工对象
    Employee employee1 = new Employee("张三");
    Employee employee2 = new Employee("李四");
    Employee employee3 = new Employee("王五");
    //设置关联关系
    //给超市管理系统分配员工
    project1.getEmployees().add(employee1);
    project1.getEmployees().add(employee2);
    project1.getEmployees().add(employee3);
    //给酒店管理系统分配员工
    project2.getEmployees().add(employee1);
    project2.getEmployees().add(employee2);
    //保存
    projectRepository.save(project1);
    projectRepository.save(project2);
}
```

注意：必须在事务环境中运行，否则会出现 `detached entity passed to persist` 错误