## 01-回顾与今日内容介绍 03:27

|  |
| --- |
| 第一 Specification动态查询  第二 多表之间的关系和操作多表的操作步骤  第三 完成多表操作  i.一对多操作  ii.多对多操作 |
|  |

## 02-动态查询的概述 03:24

|  |
| --- |
| 实际开发过程中需要我们需要根据条件动态的生成sql 语句如果每一种查询都写一个方法会是代码非常的难以维护,所以我们需要一个功能能动态的生成sql 语句 |

## 03-JpaSpecificationExecutor方法介绍 08:23

|  |
| --- |
| JpaSpecificationExecutor 方法列表    T findOne(Specification<T> spec); //查询单个对象  List<T> findAll(Specification<T> spec); //查询列表  //查询全部，分页  //pageable：分页参数  //返回值：分页pageBean（page：是springdatajpa提供的）  Page<T> findAll(Specification<T> spec, Pageable pageable);  //查询列表  //Sort：排序参数  List<T> findAll(Specification<T> spec, Sort sort);  long count(Specification<T> spec);//统计查询    \* Specification ：查询条件  自定义我们自己的Specification实现类  实现  //root：查询的根对象（查询的任何属性都可以从根对象中获取）  //CriteriaQuery：顶层查询对象，自定义查询方式（了解：一般不用）  //CriteriaBuilder：查询的构造器，封装了很多的查询条件  Predicate toPredicate(Root<T> root, CriteriaQuery<?> query, CriteriaBuilder cb); //封装查询条件 |

## 04-搭建测试环境 06:18

|  |
| --- |
|  |

## 05-构造查询条件的步骤分析 04:39

|  |
| --- |
| *1.实现Specification接口（提供泛型：查询的对象类型） 2.实现toPredicate方法（构造查询条件） 3.需要借助方法参数中的两个参数（  root：获取需要查询的对象属性  CriteriaBuilder：构造查询条件的，内部封装了很多的查询条件（模糊匹配，精准匹配）  ）* |

补充:

|  |
| --- |
| 我们也可以不使用匿名内部类的方式进行  例如:我们自己创建一个匿名内部类  **public class** MySpecification **implements** Specification<Customer> {  @Override  **public** Predicate toPredicate(Root<Customer> root, CriteriaQuery<?> query, CriteriaBuilder cb) {  **return null**;  } }  我们的测试方法可以这么写  @Test **public void** testSpec10() {  Specification spec =**new** MySpecification();  **customerDao**.findAll(spec);  } |

## 06-使用动态查询完成查询单个对象 08:44

|  |
| --- |
| 通过 Path<Object> custName = root.get(**"custId"**) 获取属性  通过CriteriaBuilder 对象 构造查询条件  cb.equal(custName, **"传智播客"**); |
| 注意:  1是equal 方法不是equals 方法  2 return null 不会报空指针异常,但是会告诉SpringDataJPA 没有查询条件,结果是: 自动生成的sql 没有where 条件,导致查询出全部数据  3,Root: 获取对象属性,用来拼接条件  4,CriteriaBuilder：构造查询条件的，内部封装了很多的查询条件（模糊匹配，精准匹配） |

## 07-动态查询完成多条件拼接 12:18

|  |
| --- |
| 多个条件需要用 CriteriaBuilder 对象的方法进行拼接  cb.and(...)  cb.or(...) |
|  |

## 08-模糊匹配查询列表 07:40

|  |
| --- |
| *equal ：直接的到path对象（属性），然后进行比较即可 gt，lt,ge,le,like : 得到path对象，根据path指定比较的参数类型，再去进行比较  指定参数类型：path.as(类型的字节码对象)*  例如  Predicate like = cb.like(custName.as(String.**class**), **"传智播客%"**); |

## 09-排序 04:39

|  |
| --- |
| Sort sort = **new** Sort(Sort.Direction.***DESC***,**"custId"**);   1. 创建一个排序对象,然后在查询方法中传递排序对象 |
| 补充:  如果有多个排序字段,Sort对象的构造函数接受的是可变参数列表,我们可以传入多个字段 |
| //多字段排序  Sort sort = **new** Sort(Sort.Direction.***DESC***,**"custId"**); Sort sort1 = **new** Sort(Sort.Direction.***ASC***,**"custNmae"**); Sort and = sort1.and(sort); |

## 10-分页 12:07

|  |
| --- |
| *1我们需要传递一个Pageable 对象*  *2PageRequest对象是Pageable接口的实现类*  */\*\*  \* 创建PageRequest的过程中，需要调用他的构造方法传入两个参数  \* 第一个参数：当前查询的页数（从0开始）  \* 第二个参数：每页查询的数量  \*/*  3返回Page 对象 page.getContent()*//得到数据集合列表* page.getTotalElements()*//得到总条数* page.getTotalPages()*//得到总页数* |
| *补充:*  *如果我们向排序后分页查询:*  *PageRequest 对象有多个构造函数*  *其中一个就是可以接受一个Sort 对象 如下:*  **public** PageRequest(**int** page, **int** size, Sort sort) {  **super**(page, size);  **this**.**sort** = sort; } |
| 先排序在分页  Sort sort = **new** Sort(Sort.Direction.***DESC***,**"custId"**); Pageable pageable = **new** PageRequest(2,3,sort); |

## 11-specification查询总结 02:13

|  |
| --- |
|  |

## 12-多表关系回顾 06:59

|  |
| --- |
| 一对一  一对多：  一的一方：主表  多的一方：从表  外键：需要再从表上新建一列作为外键，他的取值来源于主表的主键  多对多：  中间表：中间表中最少应该由两个字段组成，这两个字段做为外键指向两张表的主键，又组成了联合主键 |
| 例如:  讲师对学员：一对多关系  用户/角色: 多对多的关系 |
| 包含关系：  可以通过实体类中的包含关系描述表关系  例如:  Teacher{  private List<Student> students  }  Student{  private Teacher teacher;  } |
|  |

## 13-多表操作的分析步骤 03:06

|  |
| --- |
| 1.明确表关系  2.确定表关系（描述 外键|中间表）  3.编写实体类，再实体类中描述表关系（包含关系）  4.配置映射关系 |

## 14-一对多：案例分析 06:25

|  |
| --- |
| 客户:--- 公司  联系人:---- 员工  一个公司有多个员工,一个员工只能属于一家公司 |
| 分析步骤  1.明确表关系  一对多关系  2.确定表关系（描述 外键|中间表）  主表：客户表  从表：联系人表  \* 再从表上添加外键  3.编写实体类，再实体类中描述表关系（包含关系）  客户：再客户的实体类中包含一个联系人的集合  联系人：在联系人的实体类中包含一个客户的对象  4.配置映射关系  \* 使用jpa注解配置一对多映射关系 |

## 15-一对多：单表操作测试环境搭建 11:40

|  |
| --- |
| 一对多的环境是基于两个单表的,所以我们先搭建 两个单独的表对象的环境  1)这里需要创建两个对象的domian,  2)创建2个Dao 接口并继承 JpaRepository,JpaSpecificationExecutor |

## 16-一对多：配置一对多和多对一 12:22

|  |
| --- |
| 我们在一对一的基础上搭建环境: |
| 多对一  @ManyToOne(targetEntity = Customer.**class**) @JoinColumn(name = **"lkm\_cust\_id"**,referencedColumnName = **"cust\_id"**) **private** Customer **customer**; |
| *一对多*  @OneToMany*(*targetEntity =LinkMan*.***class***)*  @JoinColumn(name = **"lkm\_cust\_id"**,referencedColumnName = **"cust\_id"**) |
| *说明*  1)@ManyToOne : 配置多对一关系(配置”多”的一方)  2)@OneToMany : 配置一对多关系 (配置”一”的一方)  3)targetEntity：对方的实体类字节码  4)name = **"lkm\_cust\_id" :** 指的是 从表的外键字段名称  5) referencedColumnName = **"cust\_id"** 主表的主键字段名称 |

我们也可以使用Hibernate 特有的建表配置方式

|  |
| --- |
| *<!--注入jpa的配置信息  加载jpa的基本配置信息和jpa实现方式（hibernate）的配置信息  hibernate.hbm2ddl.auto : 自动创建数据库表  create ： 每次都会重新创建数据库表  update：有表不会重新创建，没有表会重新创建表 -->* <**property name="jpaProperties"** >  <**props**>  <**prop key="hibernate.hbm2ddl.auto"**>update</**prop**> |

## 17-一对多：保存操作-上 12:03

|  |
| --- |
| 创建的两个对象需要通过set 建立关系 |

## 18-一对多：放弃外键维护 10:08

|  |
| --- |
| 1. 遵循数据库外键的维护策略: 我们数据库维护是多的一方在维护外键关系,所以我们这里也用多的一方维护外键关系 2. 多的一方维护外键 比少的一方维护外键, 少一条update 语句,提升程序效率 |
| 放弃外键的方式:  在”一”的一方的注解改为  @OneToMany(mappedBy = **"customer")**  说明:  *mappedBy：对方配置关系的属性名称* |
| 放弃外键的一方在保存时将不会再维护外键 |

## 19-一对多：删除的说明和级联的引入 05:27

|  |
| --- |
| *cascade : 配置级联（可以配置到设置多表的映射关系的注解上）  CascadeType.all : 所有  MERGE ：更新  PERSIST ：保存  REMOVE ：删除* |

## 20-一对多：级联操作 08:40

|  |
| --- |
| 注意:   1. 级联配置在操作主体的一方 2. 在实际开发中，级联请慎用！ |
|  |
|  |

## 21-多对多：案例以及搭建基本框架 10:19

|  |
| --- |
| 1. 多对多也是在两个单表基础上搭建起来的   所以还是一样搭建多对多的环境 |

## 22-多对多：映射配置 09:49

|  |
| --- |
| ***@ManyToMany(targetEntity*** *= Role.class) //多对多*  *targetEntity：代表对方的实体类字节码*  ***@JoinTable***  *name : 中间表的名称*  *joinColumns：配置当前对象在中间表的外键字段名称*  ***@JoinColumn的数组***  *name：外键所在的中间表字段名称*  *referencedColumnName：参照的主表的主键字段名称名*  *inverseJoinColumns，对方对象在中间表的外键字段名称* |
| @ManyToMany(targetEntity = Role.**class**) @JoinTable(name = **"sys\_user\_role"**,  *//joinColumns,当前对象在中间表中的外键表自u但* joinColumns = {@JoinColumn(name = **"sys\_user\_id"**,referencedColumnName = **"user\_id"**)},  *//inverseJoinColumns，对方对象在中间表的外键字段* inverseJoinColumns = {@JoinColumn(name = **"sys\_role\_id"**,referencedColumnName = **"role\_id"**)} ) **private** Set<Role> **roles** = **new** HashSet<Role>(); |

## 23-多对多：放弃维护权 11:34

|  |
| --- |
| 多对多是通过中见表插入数据维护的  如果两个对象双方都维护外键关系则回往中间重复插入sql 语句,导致主键冲突  解决方案是其中一方放弃维护权(被选择一方放弃维护权,数据多的一方维护) |
| *//配置多对多* @ManyToMany(mappedBy = **"roles"**) *//配置多表关系* **private** Set<User> **users** = **new** HashSet<User>(); |

## 24-多对多：级联操作 04:49

|  |
| --- |
| 和一对多一样  注意: 级联配置在操作主体的一方  在实际开发中，级联删除请慎用！ |

## 25-对象导航查询介绍 07:32

|  |
| --- |
| 1)是SpringDataJPA 默认支持的功能,无需其他配置  2)要求属性之间有外键关联关系,否则无法查询实现对象导航查询 |

## 26-对象导航查询-延迟加载 06:59

|  |
| --- |
| 从少的一方查询 默认是延迟加载  修改方法  @OneToMany(mappedBy = **"customer"**,cascade = CascadeType.***ALL***,fetch = FetchType.***EAGER***) **private** Set<LinkMan> **linkMans** = **new** HashSet<LinkMan>(); |

## 27-对象导航查询：从多方查询一方 05:39

|  |
| --- |
| 从多的一方查询 默认是立即加载 |

## 28-对象导航查询的总结 02:45

|  |
| --- |
| 从少的一方查询 默认是延迟加载  从多的一方查询 默认是立即加载 |