### Springboot整合jpa

Spring Data:起始就是spring提供的一个操作数据的框架，而Spring Data jpa只是Spring Data框架下的一个机遇JPA标准照做数据的木块。

Spring Data JPA:基于JPA的标准对数据进行操作。简化操作持久层的代码，只需要编写接口就可以了

启用spring data jpa模块中的正向工程Application.properties

*######################################################## ### Java Persistence Mysql Api ########################################################  
# Specify the DBMS***spring.jpa.database** = *MYSQL  
# Show or not log for each sql query 显示jpasql***spring.jpa.show-sql** = **true***# Hibernate ddl auto (create, create-drop, update)***spring.jpa.hibernate.ddl-auto** = **update***# Naming strategy***spring.jpa.hibernate.naming-strategy** = **org.hibernate.cfg.ImprovedNamingStrategy***# stripped before adding them to the entity manager)***spring.jpa.properties.hibernate.dialect** = **org.hibernate.dialect.MySQL5Dialect**

Orcle配置

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect

#### 注意事项

1. 由于springboot和jpa版本不同，导致@Query写法不同，有些不支持预编译的语句(sql中存在?占位符)

在使用jpa选在的注解包大多数以javax.persistence开头

1. @Query等注解sql的文档地址https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.at-query

@Query(**"from Users where name = ?1"**)  
List<Users> queryByNameUseHQL(String name);

@Query(**"from Users where name = ?1"**) 文号后面添加数字，标识方法中第几个参数

1. 在@Query中更新参数需要添加@Modifying注解

@Query(**"update Users set name = ?1 where id = ?2"**)  
@Modifying *//需要执行一个更新操作***void** updateUsersNameById(String name, Integer id);

1. 利用方法的名称必须要遵循驼峰式命名规则。findBy(关键字)+属性名称(首字母要大写)+查询条件(首字母大写)，eg:

List<Users> findByName(String name);  
  
List<Users> findById(Integer id);  
  
List<Users> findByNameAndAge(String name, Integer age);  
  
List<Users> findByNameLike(String name);

下面这个表是findBysql常用的关键字

| **Keyword** | **Sample** | **JPQL snippet** |
| --- | --- | --- |
| And | findByLastnameAndFirstname | … where x.lastname = ?1 and x.firstname = ?2 |
| Or | findByLastnameOrFirstname | … where x.lastname = ?1 or x.firstname = ?2 |
| Is,Equals | findByFirstname,findByFirstnameIs,findByFirstnameEquals | … where x.firstname = ?1 |
| Between | findByStartDateBetween | … where x.startDate between ?1 and ?2 |
| LessThan | findByAgeLessThan | … where x.age < ?1 |
| LessThanEqual | findByAgeLessThanEqual | … where x.age <= ?1 |
| GreaterThan | findByAgeGreaterThan | … where x.age > ?1 |
| GreaterThanEqual | findByAgeGreaterThanEqual | … where x.age >= ?1 |
| After | findByStartDateAfter | … where x.startDate > ?1 |
| Before | findByStartDateBefore | … where x.startDate < ?1 |
| IsNull | findByAgeIsNull | … where x.age is null |
| IsNotNull,NotNull | findByAge(Is)NotNull | … where x.age not null |
| Like | findByFirstnameLike | … where x.firstname like ?1 |
| NotLike | findByFirstnameNotLike | … where x.firstname not like ?1 |
| StartingWith | findByFirstnameStartingWith | … where x.firstname like ?1 (parameter bound with appended %) |
| EndingWith | findByFirstnameEndingWith | … where x.firstname like ?1 (parameter bound with prepended %) |
| Containing | findByFirstnameContaining | … where x.firstname like ?1 (parameter bound wrapped in %) |
| OrderBy | findByAgeOrderByLastnameDesc | … where x.age = ?1 order by x.lastname desc |
| Not | findByLastnameNot | … where x.lastname <> ?1 |
| In | findByAgeIn(Collection<Age> ages) | … where x.age in ?1 |
| NotIn | findByAgeNotIn(Collection<Age> ages) | … where x.age not in ?1 |
| True | findByActiveTrue() | … where x.active = true |
| False | findByActiveFalse() | … where x.active = false |
| IgnoreCase | findByFirstnameIgnoreCase | … where UPPER(x.firstame) = UPPER(?1) |

1. 测试时应该注意，有些JPA接口是没有启用事物的，所以要在测试注解后面增加Transactional 和Rollback，但是有一些接口是存在Transactional的注解不需要添加。版本不同这些接口中的注解也会有区别，在写测试的时候需要注意。

@Test  
@Transactional *//@Transactional与@Test 一起使用时 事务是自动回滚的。*@Rollback(**false**) *//取消自动回滚***public void** testUpdateUsersNameById() {  
 **this**.**usersRepositoryQueryAnnotation**.updateUsersNameById(**"张三三"**, 1);  
}

1. 排序分页*PagingAndSortingRepository排序分页接口*

*/\*\*  
 \* PagingAndSortingRepository 排序测试  
 \*/*@Test  
**public void** testPagingAndSortingRepositorySort() {  
 *//Order 定义排序规则* Order order = **new** Order(Direction.***DESC***,**"id"**);  
 *//Sort对象封装了排序规则* Sort sort = **new** Sort(order);  
 List<Users> list = (List<Users>)**this**.**usersRepositoryPagingAndSorting**.findAll(sort);  
 **for** (Users users : list) {  
 System.***out***.println(users);  
 }  
}  
  
*/\*\*  
 \* PagingAndSortingRepository 分页测试  
 \*/*@Test  
**public void** testPagingAndSortingRepositoryPaging() {  
 *//Pageable:封装了分页的参数，当前页，每页显示的条数。注意：他的当前页是从0开始。  
 //PageRequest(page,size) page:当前页。size:每页显示的条数* Pageable pageable = **new** PageRequest(1, 2);  
 Page<Users> page = **this**.**usersRepositoryPagingAndSorting**.findAll(pageable);  
 System.***out***.println(**"总条数："**+page.getTotalElements());  
 System.***out***.println(**"总页数"**+page.getTotalPages());  
 List<Users> list = page.getContent();  
 **for** (Users users : list) {  
 System.***out***.println(users);  
 }  
}  
  
*/\*\*  
 \* PagingAndSortingRepository 排序+分页  
 \*/*@Test  
**public void** testPagingAndSortingRepositorySortAndPaging() {  
   
 Sort sort = **new** Sort(**new** Order(Direction.***DESC***, **"id"**));  
   
 Pageable pageable = **new** PageRequest(1, 2, sort);  
   
 Page<Users> page = **this**.**usersRepositoryPagingAndSorting**.findAll(pageable);  
 System.***out***.println(**"总条数："**+page.getTotalElements());  
 System.***out***.println(**"总页数"**+page.getTotalPages());  
 List<Users> list = page.getContent();  
 **for** (Users users : list) {  
 System.***out***.println(users);  
 }  
}  
  
*/\*\*  
 \* JapRepository 排序测试  
 \*/*@Test  
**public void** testJpaRepositorySort() {  
 *//Order 定义排序规则* Order order = **new** Order(Direction.***DESC***,**"id"**);  
 *//Sort对象封装了排序规则* Sort sort = **new** Sort(order);  
 List<Users> list = **this**.**usersRepository**.findAll(sort);  
 **for** (Users users : list) {  
 System.***out***.println(users);  
 }  
}

1. 在使用jpa等一些注解的时候要注意开启对应的注解才行
2. Springboot JPA可以实现一对多，多对一，多对多的查询关系但是需要使用对应的注解

@ManyToOne(cascade=CascadeType.***PERSIST***)  
*//@JoinColumn:维护外键*@JoinColumn(name=**"roles\_id"**)  
**private** Roles **roles**;

用户同角色的一对多的关系只需要在一个类上配置就可以实现。cascade=CascadeType.***PERSIST***这段代码是实现连锁查询的。

1. 多对多配置，多对多需要对两个关联类都进行注解配置

@ManyToMany(cascade=CascadeType.***PERSIST***,fetch=FetchType.***EAGER***)  
*//@JoinTable:映射中间表  
//joinColumns:当前表中的主键所关联的中间表中的外键字段*@JoinTable(name=**"t\_roles\_menus"**,joinColumns=@JoinColumn(name=**"role\_id"**),inverseJoinColumns=@JoinColumn(name=**"menu\_id"**))  
**private** Set<Menus> **menus** = **new** HashSet<>();

@ManyToMany(mappedBy=**"menus"**)  
**private** Set<Roles> **roles** = **new** HashSet<>();

mappedBy的意思就是“被映射”,即mappedBy这方不用管关联关系,关联关系交给另一方处理