<http://blog.csdn.net/linzhiqiang0316/article/details/52639265>

版权声明：本文为博主原创文章，未经博主允许不得转载。

## 今天给大家介绍一下SpringBoot中JPA的一些常用操作，例如：增删改查、分页、排序、事务操作等功能。

**下面先来介绍一下JPA中一些常用的查询操作：**

**[java]** [view plain](http://blog.csdn.net/linzhiqiang0316/article/details/52639265) [copy](http://blog.csdn.net/linzhiqiang0316/article/details/52639265)

1. //And --- 等价于 SQL 中的 and 关键字，比如 findByHeightAndSex(int height,char sex)；
2. **public** List<User> findByHeightAndSex(**int** height,**char** sex);
4. // Or --- 等价于 SQL 中的 or 关键字，比如 findByHeightOrSex(int height,char sex)；
5. **public** List<User> findByHeightOrSex(**int** height,**char** sex);
7. //Between --- 等价于 SQL 中的 between 关键字，比如 findByHeightBetween(int min, int max)；
8. **public** List<User> findByHeightBetween(**int** min,**int** max);
10. //LessThan --- 等价于 SQL 中的 "<"，比如 findByHeightLessThan(int max)；
11. **public** List<User> findByHeightLessThan(**int** max);
13. //GreaterThan --- 等价于 SQL 中的">"，比如 findByHeightGreaterThan(int min)；
14. **public** List<User> findByHeightGreaterThan(**int** min);
16. //IsNull --- 等价于 SQL 中的 "is null"，比如 findByNameIsNull()；
17. **public** List<User> findByNameIsNull();
19. //IsNotNull --- 等价于 SQL 中的 "is not null"，比如 findByNameIsNotNull()；
20. **public** List<User> findByNameIsNotNull();
22. //NotNull --- 与 IsNotNull 等价；
23. **public** List<User> findByNameNotNull();
25. //Like --- 等价于 SQL 中的 "like"，比如 findByNameLike(String name);
26. **public** List<User> findByNameLike(String name);
28. //NotLike --- 等价于 SQL 中的 "not like"，比如 findByNameNotLike(String name)；
29. **public** List<User> findByNameNotLike(String name);
31. //OrderBy --- 等价于 SQL 中的 "order by"，比如 findByNameNotNullOrderByHeightAsc()；
32. **public** List<User>findByNameNotNullOrderByHeightAsc();
34. //Not --- 等价于 SQL 中的 "！ ="，比如 findByNameNot(String name)；
35. **public** List<User> findByNameNot(String name);
37. //In --- 等价于 SQL 中的 "in"，比如 findByNameIN(String name);
38. **public** List<User> findByNameIn(String name);
40. //NotIn --- 等价于 SQL 中的 "not in"，比如 findByNameNotIN(String name);
41. **public** List<User> findByNameNotIn(String name);

JPA中的风格就是这样，每个方法其实都是一条SQl命令，通过一些关键字就可以实现SQL中类似于like in等等之类的命令了。

最重要的是我们再开发的过程中，只需要编写dao中一个个方法，不需要我们编写dao的实现类，这样就可以大大的挺高代码的复用率、提高我们的开发效率。

说道这里不免会有人会问，那一些比较复杂的关联查询要怎么实现呢，JPA的处理方法是：利用原生的SQl命令来实现那些复杂的关联查询，下面就来看下案例。

**[java]** [view plain](http://blog.csdn.net/linzhiqiang0316/article/details/52639265) [copy](http://blog.csdn.net/linzhiqiang0316/article/details/52639265)

1. //利用原生的SQL进行查询操作
2. @Query(value = "select o.\* from orders o ,user u where o.uid=u.id and u.name=?1", nativeQuery = **true**)
3. @Modifying
4. **public** List<Order> findOrderByName(String name);
6. //利用原生的SQL进行删除操作
7. @Query(value = "delete from orders where id=?1 ", nativeQuery = **true**)
8. @Modifying
9. **public** **void** deleteOrderById(**int** id);
11. //利用原生的SQL进行删除操作
12. @Query(value = "delete from orders where uid=?1 ", nativeQuery = **true**)
13. @Modifying
14. **public** **void** deleteOrderByUId(**int** uid);
16. //利用原生的SQL进行修改操作
17. @Query(value = "update orders set name=?1 where id=?2 ", nativeQuery = **true**)
18. @Modifying
19. **public** **void** updateOrderName(String name,**int** id);
21. //利用原生的SQL进行插入操作
22. @Query(value = "insert into orders(name,uid) value(?1,?2)", nativeQuery = **true**)
23. @Modifying
24. **public** **void** insertOrder(String name,**int** uid);

上面的案例中给出了，利用JPA实现原生的SQL操作，可以很方便的进行数据库表的操作。  
所以如果是那种查询语句不是非常复杂，对查询时间要求不是特别苛刻的项目，完全可以采用JPA来进行项目的开发。

**下面接着来介绍JPA是怎么实现分页的效果，其实JPA脱胎于hibernate，所以本身就对分页功能有很好的支持。下面给出具体例子：**

**[java]** [view plain](http://blog.csdn.net/linzhiqiang0316/article/details/52639265) [copy](http://blog.csdn.net/linzhiqiang0316/article/details/52639265)

1. //实现分页功能
2. Page<User> findByNameNot(String name,Pageable pageable);

**[java]** [view plain](http://blog.csdn.net/linzhiqiang0316/article/details/52639265) [copy](http://blog.csdn.net/linzhiqiang0316/article/details/52639265)

1. @RequestMapping(value = "/params", method= RequestMethod.GET)
2. @ResponseBody
3. **public** String getEntryByParams(@RequestParam(value = "name", defaultValue = "林志强") String name, @RequestParam(value = "page", defaultValue = "0") Integer page, @RequestParam(value = "size", defaultValue = "15") Integer size) {
4. Sort sort = **new** Sort(Sort.Direction.DESC, "id");
5. Pageable pageable = **new** PageRequest(page, size, sort);
6. Page<User> pages=userDao.findByNameNot(name,pageable);
7. Iterator<User> it=pages.iterator();
8. **while**(it.hasNext()){
9. System.out.println("value:"+((User)it.next()).getId());
10. }
11. **return** "success...login....";
12. }

上面的代码一个是在dao层中的，一个是在controller中的。

dao层中添加一个返回值为Page，参数值为Pageable。controller层中通过实例化Pageable这个类，然后调用dao层这个分页方法。

通过这些步骤就可以轻轻松松的实现分页的效果啦，看起来是不是特别方便。

**最后在给大家介绍一下JPA是如何实现事务操作的。其实因为SpringBoot中已经对事务做了很好的封装了，使用起来特别方便。下面看一下案例：**

**[java]** [view plain](http://blog.csdn.net/linzhiqiang0316/article/details/52639265) [copy](http://blog.csdn.net/linzhiqiang0316/article/details/52639265)

1. @RequestMapping("/saveOrder")
2. @ResponseBody
3. @Transactional()
4. **public** String saveOrder(){
5. Order o1=**new** Order("11",2);
6. Order o2=**new** Order("22",2);
7. Order o3=**new** Order("33",2);
8. Order o4=**new** Order("44",2);
9. orderDao.save(o1);
10. orderDao.save(o2);
11. orderDao.save(o3);
12. orderDao.save(o4);
13. **return** "successfull....saveOrder......";
14. }

只要在方法的上面加上**@Transaction** 这个注解就可以轻轻松松的实现事务的操作了，是不是特别方便啊。

不过这里有几点需要注意的是：

1.这个注解实现的事务管理器是默认的，如果不想要默认是事务管理器，可以自己进行添加，我这里就不多介绍了。

2.事务的隔离级别也是可以自己设置的，具体请看这篇博客 [点击打开链接](http://blog.csdn.net/linzhiqiang0316/article/details/52638039)

3.事务的传播行为也是可以自己设置的，具体请看这篇博客[点击打开链接](http://blog.csdn.net/linzhiqiang0316/article/details/52638039)  
  
到此到此关于JPA的一些常规方法就介绍完了，谢谢大家的阅读。

**如果大家想要源代码或者对博客有啥异议都可以加我QQ：208017534  欢迎打扰哦！！！**