JPA查询方法

方法命名查询

概述

> 顾名思义，方法命名规则查询就是根据方法的名字，就能创建查询。只需要按照Spring Data JPA提供的方法命名规则定义方法的名称，就可以完成查询工作。Spring Data JPA在程序执行的时候会根据方法名称进行解析，并自动生成查询语句进行查询
>
> 按照Spring Data JPA 定义的规则，查询方法以 `findBy` 开头，涉及条件查询时，条件的属性用条件关键字连接，要注意的是：**条件属性首字母需大写。**框架在进行方法名解析时，会先把方法名多余的前缀截取掉，然后对剩下部分进行解析。

关键字

Keyword	**Sample**	**JPQL**
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	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 ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection age)	... 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)

案例：方法命名查询

环境准备

> 需求：以User类为示例，对User类进行查询操作
>
> 步骤：
>
> 1.定义User实体类
>
> 2.定义UserRepository接口
>
> 3.测试方法命名查询

实体类

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

import javax.persistence.*;

@Entity//标识是一个实体类
@Table(name="t_user")//表名
public class User {

    @Id//主键
    @GeneratedValue(strategy=GenerationType.IDENTITY)//主键自增类型
    //当属性名与列名一致时，可以省略@Column
    @Column
    private Integer id;
    private String userName;
    private Integer age;
    private String address;
```

```
    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 Integer getAge() {
        return age;
    }
    public void setAge(Integer age) {
        this.age = age;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", userName=" + userName + ", age=" + age + ", address=" + address +
"]";
    }

    public User() {
    }

    public User(String userName, Integer age, String address) {
        this.userName = userName;
        this.age = age;
        this.address = address;
    }

    public User(String userName, Integer age, String address,Integer id) {
        this.userName = userName;
        this.age = age;
        this.address = address;
        this.id = id;
    }
}
```

repository接口

```
package com.kazu.springdatajpa02.dao;

import com.kazu.springdatajpa02.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface UserRepository extends JpaRepository<User,Integer> {

    //注意：方法的名称必须要遵循驼峰式命名规则。xxxBy(关键字)+属性名称(首字母大写)+查询条件(首字母大写)

    /**
     * 根据用户名精确查询
     * @param userName
     * @return
     */
    List<User> findByUserName(String userName);

    /**
     * 模糊查询
     * @param userName
```

```
     * @return
     */
    List<User> queryByUserNameLike(String userName);

    /**
     * 根据用户名和地址模糊查询
     * @param userName
     * @param address
     * @return
     */
    List<User> findByUserNameLikeAndAddressLike(String userName,String address);
}
```

> 注意：方法名称除了find的前缀之外，还有以下前缀，具体可查看PartTree类的源码（快速搜索查看类快捷键：ctrl + shift + T）
>
> PartTree类源码如下：

```
public class PartTree implements Streamable<PartTree.OrPart> {
    private static final String KEYWORD_TEMPLATE = "(%s)(?=(\\p{Lu}|\\P{InBASIC_LATIN}))";
    private static final String QUERY_PATTERN = "find|read|get|query|stream";
    private static final String COUNT_PATTERN = "count";
    private static final String EXISTS_PATTERN = "exists";
    private static final String DELETE_PATTERN = "delete|remove";
    private static final Pattern PREFIX_TEMPLATE =
Pattern.compile("^(find|read|get|query|stream|count|exists|delete|remove)((\\p{Lu}.*?))??By");
    private final PartTree.Subject subject;
    private final PartTree.Predicate predicate;
```

测试类

```
package com.kazu.springdatajpa02;

import com.kazu.springdatajpa02.dao.UserRepository;
import com.kazu.springdatajpa02.entity.User;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;
import java.util.List;

@RunWith(SpringRunner.class)
@SpringBootTest
public class Springdatajpa02ApplicationTests {

    @Resource
    private UserRepository userRepository;

    @Test
    public void test1() {
       List<User> users = userRepository.findByUserName("张三");
        for (User user : users) {
            System.out.println(user);
        }
    }

    @Test
    public void test2() {
        List<User> users = userRepository.queryByUserNameLike("李%");
        for (User user : users) {
            System.out.println(user);
        }
    }

    @Test
    public void test3() {
        List<User> users = userRepository.findByUserNameLikeAndAddressLike("李%","%海珠%");
```

```
        for (User user : users) {
            System.out.println(user);
        }
    }

}
```

小结

1. 方法名称必须要遵循驼峰式命名规则。xxxBy(**关键字**)+**属性名称(首字母大写)**+**查询条件**+(**首字母大写**)+**查询条件**....
2. 方法前缀可以为 `find` 、 `read` 、 `get` 、 `query` 、 `stream` 、 `count` 、 `exists` 、 `remove` 、 `delete` 具体查看 `PartTree` 类

注解式查询方法

概述

使用Spring Data JPA提供的查询方法已经可以解决大部分的应用场景，但是对于某些业务来说，我们还需要灵活的构造查询条件，这时就可以使用@Query注解，结合JPQL的语句方式完成查询。

@Query详解

语法及源码

```
public @interface Query {
    /**
    * 指定JPQL的查询语句。（nativeQuery属性为true时，为原生SQL语句）
    */
    String value() default "";

    /**
    * 指定count的JPQL语句。（nativeQuery属性为true时，为原生SQL语句）
    */
    String countQuery() default "";

    /**
    * 根据哪个字段类count，一般默认即可
    */
    String countProjection() default "";

    /**
    * 默认为false，表示value属性里面的是JPQL语句，不是原生SQL语句
    */
    boolean nativeQuery() default false;

    /**
    * 可以制定一个query的名字，名称必须唯一
    如果不指定，默认的生成规则是：{$domainClass}.${queryMethodName}
    */
    String name() default "";
   /*
    * 可以指定一个count的query的名字，必须唯一的。
    * 如果不指定，默认的生成规则是：
    * {$domainClass}.${queryMethodName}.count
    */
    String countName() default "";
}
```

@Query用法

> 使用命名查询为实体声明查询是一种有效的方法，对于少量查询很有效。一般只需要关心 @Query 里面的 value 和 nativeQuery 的值。使用声明式 JPQL 查询有个好处，就是启动的时候就知道你的语法正确不正确。

案例1-JPQL查询

> 使用JPQL语句完成单条件和多条件查询

接口

```
/**
 * 根据用户名模糊查询
 * @param username  用户名
 * @return
 */
@Query("from User where userName like ?1%")
List<User> findUserListByName(String username);

/**
 * 根据用户名和地址模糊查询
 * @param username  用户名
 * @param address   地址
 * @return
 */
@Query("from User where userName like ?1% and address like %?2%")
List<User> findUserList(String username,String address);
```

> **注意：参数下标从1开始，占位符要与参数位置正确匹配**

测试类

```
@Test
public void testJPQL1(){
    List<User> users = userRepository.findUserListByName("李");
    for (User user : users) {
        System.out.println(user);
    }
}

@Test
public void testJPQL2(){
    List<User> users = userRepository.findUserList("李","海珠");
    for (User user : users) {
        System.out.println(user);
    }
}
```

案例2-原生SQL语句

接口

```
    /**
     * 原生SQL查询
     * @param username  用户名
     * @param address   地址
     * @return
     */
    @Query(value = "select * from - where user_name like %?1% and address like %?2%",nativeQuery =
true)
    List<User> findUserListBySQL(String username,String address);
```

注意：使用原生SQL，需要将 `nativeQuery` 属性设置为 `true`

测试类

```
@Test
public void testSQL(){
    List<User> users = userRepository.findUserListBySQL("李","海珠");
    for (User user : users) {
        System.out.println(user);
    }
}
```

@Param用法

概述

默认情况下，参数是通过顺序绑定在查询语句上的，这使得查询方法对参数位置的重构容易出错。为了解决这个问题，可以使用 `@Param` 注解指定方法参数的具体名称，通过绑定的参数名字做查询条件，这样不需要关心参数的顺序，推荐这种做法，比较利于代码重构

@Param的使用

接口

```
    /**
     * 命名参数查询
     * @param username
     * @param address
     * @return
     */
    @Query("from User where userName like :username% and address like %:address%")
    List<User> findUserListByParam(@Param("username") String username,@Param("address") String
address);
```

注意：

1. @Param注解必须提供且注解内容与命名参数一致
2. 命名参数定义与Hibernate一致，使用英文冒号定义

测试

```
    @Test
    public void testParam(){
        List<User> users = userRepository.findUserListByParam("李","海珠");
        for (User user : users) {
            System.out.println(user);
        }
    }
```

@Modifying修改查询之更新

接口

```
@Modifying//修改查询，标记为更新
@Query("update User set userName =?1 where id =?2")
int updateUser(String username,int id);
```

测试类

```
@Test
@Transactional//需要开启事务
public void testUpdate(){
    userRepository.updateUser("test",2);
}
```

> 注意：需要加入 `@Transactional` 注解开启事务

@Modifying修改查询之删除

接口

```
@Modifying//修改查询，标记为删除
@Query("delete from User where id =?1")
int deleteUser(int id);
```

测试类

```
@Test
@Transactional//需要开启事务
public void testDelete(){
    userRepository.deleteUser(10);
}
```

> 注意：需要加入 `@Transactional` 注解开启事务

@Query注解的优缺点与实战经验分享

优点

> 1. 可以灵活快速的使用 JPQL 和 SQL
> 2. 对返回的结果和字段可以自定义

3. 支持连表查询和对象关联查询，可以组合出来复杂的 SQL 或者 JPQL
4. 可以很好的表达你的查询思路
5. 灵活性非常强，快捷方便

缺点

1. 不支持动态查询条件，参数个数如果是不固定的不支持
2. 若将返回结果用 Map 或者 Object[] 数组接收结果，会导致调用此方法的开发人员不知道返回结果里面到底有些什么数据

实战经验

1. 当出现很复杂的 SQL 或者 JPQL 的时候建议用视图
2. 返回结果一定要用对象接收，最好每个对象里面的字段和你返回的结果一一对应
3. 能用 JPQL 的就不要用 SQL