<http://blog.csdn.net/yingxiake/article/details/51016234>

版权声明：本文为博主原创文章，未经博主允许不得转载。

介绍@Query注释之前，先看看怎么利用@NamedQuery进行命名查询

1.现在实体类上定义方法已经具体查询语句

@Entity

@NamedQuery(name = "Task.findByTaskName",

query = "select t from Task t where t.taskName = ?1")

public class Task{

}

2.然后我们继承接口之后，就可以直接用这个方法了，它会执行我们定义好的查询语句并返回结果

public interface TaskDao extends JpaRepository<Task, Long> {

Task findByTaskName(String taskName);

}

试想一下，如果我们想自己定义执行查询，利用命名查询，显然不行，因为，会在实体类上写很多的@NamedQuery，这种情况的话，我们可以用@Query直接在方法上定义查询语句，例如这样

public interface TaskDao extends JpaRepository<Task, Long> {

@Query("select t from Task t where t.taskName = ?1")

Task findByTaskName(String taskName);

}

@Query上面的1代表的是方法参数里面的顺序，除了写hql，我们还可以写sql语句

public interface TaskDao extends JpaRepository<Task, Long> {

@Query("select \* from tb\_task t where t.task\_name = ?1", nativeQuery = true)

Task findByTaskName(String taskName);

}

在参数绑定上，我们还可以这样子用

public interface TaskDao extends JpaRepository<Task, Long> {

@Query("select t from Task t where t.taskName = :taskName and t.createTime = :createTime")

Task findByTaskName(@Param("taskName")String taskName,@Param("createTime") Date createTime);

}

当然在参数绑定上，我们还可以直写问号

public interface TaskDao extends JpaRepository<Task, Long> {

@Query("select t from Task t where t.taskName = ? and t.createTime = ?")

Task findByTaskName(String taskName, Date createTime);

}

再利用SpEL表达式，我们把实体类写成动态的

public interface TaskDao extends JpaRepository<Task, Long> {

@Query("select t from #{#entityName} t where t.taskName = ? and t.createTime = ?")

Task findByTaskName(String taskName, Date createTime);

}

这个的作用就是，当俩个实体类都有共同的父类的时候，例如这样

// JPA 基类的标识

@MappedSuperclass

@SuppressWarnings("serial")

public abstract class IdEntity implements Serializable{

protected Long id;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

}

@Entity

public class Task extends IdEntity{

}

@Entity

public class Project extends IdEntity{

}

然后有一个通用的接口

public interface GenericDao<T> extends JpaRepository<T, ID> {

@Query("select t from #{#entityName} t where t.id= ?1")

public T findById(Long id);

}

再然后就taskDao和projectDao来继承这个接口，这样子的话，把公用的方法放在通用接口上，就不用重复写方法了。

好，下面再说下，利用@Modifying进行更新

@Modifying

@Query("update Task t set t.taskName = ?1 where t.id = ?2")

int updateTask(String taskName, Long id);

在这里我们说下，[**spring**](http://lib.csdn.net/base/javaee) data jpa的查询策略，spring data jpa可以利用创建方法进行查询，也可以利用@Query注释进行查询，那么如果在命名规范的方法上使用了@Query，那spring data jpa是执行我们定义的语句进行查询，还是按照规范的方法进行查询呢？看下查询策略

查询策略的配置可以在配置query-lookup-strategy，例如这样

<jpa:repositories base-package="com.liuxg.\*\*.dao"

repository-impl-postfix="Impl"

query-lookup-strategy = "create-if-not-found"

entity-manager-factory-ref="entityManagerFactory"

transaction-manager-ref="transactionManager" >

</jpa:repositories>

他有三种值可以配置

1. create-if-not-found(默认)：如果通过 @Query指定查询语句，则执行该语句，如果没有，则看看有没有@NameQuery指定的查询语句，如果还没有，则通过解析方法名进行查询
2. create：通过解析方法名字来创建查询。即使有 @Query，@NameQuery都会忽略
3. use-declared-query：通过执行@Query定义的语句来执行查询，如果没有，则看看有没有通过执行@NameQuery来执行查询，还没有则抛出异常

@Query就先看到这里，下次再了解下怎么拓展spring data jpa 接口，例如我既想用sping data jpa的接口，可是我又想自己定义一些接口，我们把他们合二为一呢？？