## 1）spring data jpa介绍

JPA（Java Persistence API）是Sun官方提出的Java持久化规范。它为Java开发人员提供了一种对象/关联映射工具来管理Java应用中的关系数据。他的出现主要是为了简化现有的持久化开发工作和整合ORM技术，结束现在Hibernate，TopLink，JDO等ORM框架各自为营的局面。值得注意的是，JPA是在充分吸收了现有Hibernate，TopLink，JDO等ORM框架的基础上发展而来的，具有易于使用，伸缩性强等优点。从目前的开发社区的反应上看，JPA收到了极大的支持和赞扬，其中就包括了Spring与EJB3.0的开发团队。

注意：JPA是一套规范，不是一套产品，那么像Hibernate，TopLink，JDO他们是一套产品，如果说这些产品实现了这个JPA规范，那么我们就可以叫他们为JPA的实现产品。

Spring Data JPA是Spring基于ORM框架、JPA规范的基础上封装的一套JPA应用框架，可使开发者用极简的代码即可实现对数据的访问和操作。

## 2）基本查询

基本查询也分两种，一种是spring data默认已经实现，一种是根据查询的方法来自动解析成SQL。

### 预先生成方法

spring data jpa默认预先生成了一些基本的CURD的方法，例如：增、删、改、查等等。

1. 继承JpaRepository

public interface UserRepository extends JpaRepository<User,Long> {

}

1. 使用默认方法

@Test

public void testBaseQuery() throws Exception {

User user=new User();

userRepository.findAll();

userRepository.findOne(1l);

userRepository.save(user);

userRepository.delete(user);

userRepository.count();

userRepository.exists(1l);

// ...

}

### 自定义简单查询

自定义的简单查询就是根据方法名来自动生成SQL，主要的语法是findXXBy，readAxxBy，queryXXBy，countXXBy，getXXBy后面跟属性名称：

User findByUserName(String userName);

也使用一些加一些关键字And、Or

User findByUserNameOrEmail(String username, String email);

修改、删除、统计也是类似语法

Long deleteById(Long id);

Long countByUserName(String userName)

基本上SQL体系中的关键词都可以使用，例如：LIKE、IgnoreCase、OrderBy。

List<User> findByEmailLike(String email);

User findByUserNameIgnoreCase(String userName);

List<User> findByUserNameOrderByEmailDesc(String email);

具体的关键字，使用方法和生产SQL如下表所示：

| **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 | 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(Collectionages) | … where x.age in ?1 |
| NotIn | findByAgeNotIn(Collectionage) | … 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) |

## 3）复杂查询

在实际的开发中我们需要用到分页、删选、连表等查询的时候就需要特殊的方法或者自定义SQL

## 4）分页查询

分页查询在实际使用中非常普遍了，spring data jpa已经帮我们实现了分页的功能，在查询的方法中，需要传入参数Pageable，当查询中有多个参数的时候Pageable建议做为最后一个参数传入：

Page<User> findALL(Pageable pageable);

Page<User> findByUserName(String userName,Pageable pageable);

Pageable是spring封装的分页实现类，使用的时候需要传入页数、每页条数和排序规则：

@Test

public void testPageQuery() throws Exception {

int page=1,size=10;

Sort sort = new Sort(Direction.DESC, "id");

Pageable pageable = new PageRequest(page, size, sort);

userRepository.findALL(pageable);

userRepository.findByUserName("testName", pageable);

}

## 5）限制查询

有时候我们只需要查询前N个元素，或者支取前一个实体。

ser findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);

## 6）自定义SQL查询

其实Springdata觉大部分的SQL都可以根据方法名定义的方式来实现，但是由于某些原因我们想使用自定义的SQL来查询，springdata也是完美支持的；在SQL的查询方法上面使用@Query注解，如涉及到删除和修改在需要加上@Modifying.也可以根据需要添加@Transactional对事物的支持，查询超时的设置等。

@Modifying

@Query("update User u set u.userName = ?1 where c.id = ?2")

int modifyByIdAndUserId(String userName, Long id);

@Transactional

@Modifying

@Query("delete from User where id = ?1")

void deleteByUserId(Long id);

@Transactional(timeout = 10)

@Query("select u from User u where u.emailAddress = ?1")

User findByEmailAddress(String emailAddress);

## 7）多表查询

多表查询在springdatajpa中有两种实现方式，第一种是利用hibernate的级联查询来实现，第二种是创建一个结果集的接口来接收连表查询后的结果，这里主要第二种方式。

首先需要定义一个结果集的接口类。

public interface HotelSummary {

City getCity();

String getName();

Double getAverageRating();

default Integer getAverageRatingRounded() {

return getAverageRating() == null ? null : (int) Math.round(getAverageRating());

}

}

查询的方法返回类型设置为新创建的接口：

@Query("select h.city as city, h.name as name, avg(r.rating) as averageRating"

- "from Hotel h left outer join h.reviews r where h.city = ?1 group by h")

Page<HotelSummary> findByCity(City city, Pageable pageable);

@Query("select h.name as name, avg(r.rating) as averageRating "

- "from Hotel h left outer join h.reviews r group by h")

Page<HotelSummary> findByCity(Pageable pageable);

使用：

Page<HotelSummary> hotels = this.hotelRepository.findByCity(new

PageRequest(0, 10, Direction.ASC, "name"));

for(HotelSummary summay:hotels){

System.out.println("Name" +summay.getName());

}

在运行中Spring会给接口（HotelSummary）自动生产一个代理类来接收返回的结果，代码汇总使用getXX的形式来获取。

## 8）多数据源的支持

### 同源数据库的多源支持

日常项目中因为使用的分布式开发模式，不同的服务有不同的数据源，常常需要在一个项目中使用多个数据源，因此需要配置spring data jpa对多数据源的使用，一般分一下为三步：

1. 配置多数据源
2. 不同源的实体类放入不同包路径
3. 声明不同的包路径下使用不同的数据源、事务支持

### 异构数据库多源支持

比如我们的项目中，即需要对mysql的支持，也需要对mongodb的查询等。

实体类声明@Entity关系型数据库支持类型、声明@Document为mongodb支持类型，不同的数据源使用不同的实体就可以了：

interface PersonRepository extends Repository<Person, Long> {

…

}

@Entity

public class Person {

…

}

interface UserRepository extends Repository<User, Long> {

…

}

@Document

public class User {

…

}

但是，如果User用户既使用mysql也使用mongodb呢，也可以做混合使用：

interface JpaPersonRepository extends Repository<Person, Long> {

…

}

interface MongoDBPersonRepository extends Repository<Person, Long> {

…

}

@Entity

@Document

public class Person {

…

}

也可以通过对不同的包路径进行声明，比如A包路径下使用mysql，B包路径下使用mongodb：

@EnableJpaRepositories(basePackages = "com.neo.repositories.jpa")

@EnableMongoRepositories(basePackages = "com.neo.repositories.mongo")

interface Configuration { }

## 9）其它

### 使用枚举

使用枚举的时候，我们希望数据库中存储的是枚举对应的String类型，而不是枚举的索引值，需要在属性上面添加@Enumerated(EnumType.STRING)注解：

@Enumerated(EnumType.STRING)

@Column(nullable = true)

private UserType type;

### 不需要和数据库映射的属性

正常情况下我们在实体类上加入注解@Entity，就会让实体类和表相关连如果其中某个属性我们不需要和数据库来关联只是在展示的时候做计算，只需要加上@Transient属性既可。

@Transient

private String userName;