SpringData前言

# 介绍

> Spring Data ： Spring 的一个子项目。用于简化数据库访问，支持NoSQL 和 关系数据存储。其主要目标是使数据库的访问变得方便快捷。

- SpringData 项目所支持 NoSQL 存储：

1. MongoDB （文档数据库）
2. Neo4j（图形数据库）
3. Redis（键/值存储）
4. Hbase（列族数据库）

- SpringData 项目所支持的关系数据存储技术：

1. JDBC
2. JPA
3. ject

**本笔记学习的是SpringData与JPA结合的知识**

# 作用

> JPA Spring Data ： 致力于减少数据访问层 (DAO) 的开发量. 开发者唯一要做的，就只是声明持久层的接口，其他都交给 Spring Data JPA 来帮你完成!
>
> 框架怎么可能代替开发者实现业务逻辑呢? 比如：当有一个 UserDao.findUserById() 这样一个方法声明，大致应该能判断出这是根据给定条件的 ID 查询出满足条件的 User 对象。Spring Data JPA 做的便是规范方法的名字，根据符合规范的名字来确定方法需要实现什么样的逻辑。

# 环境搭建

- 创建Maven工程并除了基本的**Spring骨架外与JPA规范的包外**还需依赖两个SpringData与JPA的Jar包

```
        <!--SpringData基础包-->
        <dependency>
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-commons</artifactId>
            <version>1.6.2.RELEASE</version>
        </dependency>
        <!--JPA与SpringData兼容包-->
        <dependency>
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-jpa</artifactId>
            <version>1.4.2.RELEASE</version>
        </dependency>
```

**注意SpringData与Spring骨架的其它包版本需一致**

# Repository接口

## 概述

Repository 接口是SpringData 的一个核心接口，它不提供任何方法，开发者需要在自己定义的接口中声明需要的方法。

SpringData的主要亮点就是开发者定义操作数据库Dao层接口时，无需再写出实现类，而是根据SpringData给出的接口方法名策略，根据它命名的要求，再按照开发者的需求写出接口方法即可。如：

`UserDao.findUserById()`

这样一个方法声明，大致应该能判断出这是根据给定条件的ID查询出满足条件的User对象。SpringData JPA做的便是规范方法的名字，根据符合规范的名字来确定方法需要实现什么样的逻辑。

## 声明为SpringData的Dao接口

与继承 Repository 等价的一种方式，就是在持久层接口上使用 @RepositoryDefinition 注解，并为其指定 domainClass 和 idClass 属性。如下两种方式是完全等价的

- **继承Repository接口**

```
public interface StudentRepositoryDao extends Repository<Student, Integer> {
```

其中Repository两个泛型分别为`映射实体对象`，`实体对象中的主键类型`。

- **@RepositoryDefinition**

```
@RepositoryDefinition(domainClass = Teacher.class, idClass = Integer.class)
public class TeacherRepositoryDefinition {
```

## Repository的继承体系

基础的 Repository提供了最基本的数据访问功能，其几个子接口则扩展了一些功能。它们的继承关系如下：

### Repository

仅仅是一个标识，表明任何继承它的均为仓库接口类。

### CrudRepository

继承 Repository，实现了一组 CRUD 相关的方法

### PagingAndSortingRepository

> 继承 CrudRepository，实现了一组分页排序相关的方法 ，但不能进行条件筛选。

### JpaRepository

> 继承 PagingAndSortingRepository，实现一组 JPA 规范相关的方法

### 自定义的 XxxxRepository

> 需要继承 JpaRepository，这样的 XxxxRepository 接口就具备了通用的数据访问控制层的能力。

### JpaSpecificationExecutor

> 不属于Repository体系，实现一组 JPA Criteria 查询相关的方法，如先进行条件判断后再进行分页或排序。

# 查询方式

## 概述

> SpringData中查询实体类方式有多种，其中比较能突出SpringData操作数据能力的有以下一些功能。

## SpringData 方法定义规范

### 简单条件查询

#### 规范

> 按照Spring Data 的规范，查询方法以`find` | `read` | `get` 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：条件属性以首字母大写。

**举例：**

定义一个 Entity实体类

```
classUser {
  private StringfirstName;
  private StringlastName;
}
```

使用And条件连接时，应这样写`findByLastNameAndFirstName(StringlastName,StringfirstName);`

条件的属性名称与个数要与参数的位置与个数一一对应。

### 查询方法解析流程

> 假如创建如下的查询：`findByUserDepUuid()`，框架在解析该方法时，首先剔除 `findBy`，然后对剩下的属性进行解析，假设查询实体为`Doc`。

1. SpringData方法解析中对每个条件参数进行层次划分优先条件有`1.以参数名中的出现的大写字母` `2.'_'：明确声明为级联属性`
2. 首先SpringData最想知道`UserDepUuid`哪一部分是一个类属性。**根据 POJO 规范，首字母变为小写**，则作为类属性最有可能出现的情况有，`1.Doc.userDepUuid` `2.Doc.userDep` `3.Doc.user`
3. 剩下部分作为级联属性则最有可能出现情况有，`1.Doc.userDep.Uuid` `2.Doc.user.DepUuid`
4. 其中`Doc.user.DepUuid`，由于末尾部分还存在超过一个大写的参数名，所以`DepUuid`参数名会继续解析
5. SpringData最希望它只是一个类属性，所以`Doc.user.DepUuid`在进行细划分的话，最有可能的出现是`1.Doc.user.DepUuid` `2.Doc.user.dep.uuid`
6. 若通过'_'明确指定`Uuid`参数为一个级联属性`UserDep_Uuid`，那么会先判断是否为级联属性的情况`Doc.user.dep.Uuid`,而越过先解析类属性的情况，若不是在继续判断是否为类属性的情况

**注意解析参数名为实体类的属性名而并非数据库的字段名**

## 支持的关键字

| 关键字 | 举例 | 最终转换成的JQPL语句片段 | 解释 |
| --- | --- | --- | --- |
| And | findByLastnameAndFirstname | ...where x.lastname=?1 and firstname=?2 | 和 |
| Or | findByLastnameOrFirstname | ...where x.lastname=?1 or x.firstname=?2 | 或 |
| Between | findByStartDateBetween | ...where x.startDate between ?1 and ?2 | 之间 |
| LessThan | findByAgeLessThan | ...where x.age<?1 | 小于 |
| GreaterThan | findByAgeGreaterThan | ...where x.age>?1 | 大于 |
| After | findByStartDateAfter | ...where x.startDate>?1 | 之后，判断时间类型 |
| Before | findByStartDateBefor | ...where x.startDate<?1 | 之前，判断时间类型 |
| IsNull | findByAgeIsNull | ...where x.age is null | 等于null |
| IsNotNull==NotNull | findByAge(Is)NotNull | ...where x.age not null | 不等于null |
| Like | findByFirstnameLike | ...where x.firstname like ?1 | 根据传入参数进行匹配筛选**类似**字段数据 |
| NotLike | findByFirstnameNotLike | ...where x.firstname not like ?1 | 根据传入参数进行匹配筛选**不类似**字段数据 |
| StartingWith | findByFirstnameStartingWith | ...where x.firstname not like ?1 | 匹配字段**开头**，底层转换JQPL语句实际在参数**前面**添加了'%'占位符 |

| 关键字 | 举例 | 最终转换成的JQPL语句片段 | 解释 |
| --- | --- | --- | --- |
| EndingWith | findByFirstnameEndingWith | ...where x.firstname not like ?1 | 匹配字段**末尾**，底层转换JQPL语句实际在参数**后面**追加了'%'占位符 |
| Containing | findByFirstnameContaining | ...where x.firstname not like ?1 | 匹配字段**包含**参数字符串，底层转换JQPL语句实际在参数**首尾**都添加了'%'占位符 |
| OrderBy | findByFirstnameOrerByAgeDesc | ...where x.firstname=?1 order by x.age desc | 指定字段进行排序 |
| Not | findByLastnameNot | ...where x.lastname <>?1 | |
| In | findByAgeIn(Collection age) | ...where x .age in ?1 | 筛选出指定字段**在**该参数集合条件下的数据列 |
| NotIn | findByAgeNotIn(Collection age) | ...where x.age not in ?1 | 筛选出指定字段**不在**该参数集合条件下的数据列 |
| True | findByActiveTrue() | ...where x.active=true | |
| False | findByActiveFalse() | ...where x.active=false | |

## @Query注解

有时候SpringData的方法规范让人确实不好记，且有些方法命名长度觉得稍长了些的话，可以通过@Query注解让开发者自己写查询数据的SQL语法。

### 简单条件查询

- 通过使用Springdata的方法规范命名我们查询Teacher时可以这么写

```
Student findByAge (Interge age)
```

- 同样使用@Query写的话，等价于

```
@Query(value="select t from Teacher t where t.age=?1")
public void 查询指定年龄的Teacher(Interge age)
```

### Query参数

### 参数传入方式

> 索引参数

```
@Query(value="select t from Teacher t where t.age=?1 and t.name=?2")
public void 查询指定年龄的Teacher(Interge age,String name);
```

> 命名参数

```
@Query(value="select t from Teacher t where t.age=:age and t.name=:name")
public void 查询指定年龄的Teacher(@Param("age")Interge age, @Param("name")String name);
```

## 继承Repository子接口

> Repository子接口中封装了大量平常使用的查询操作，其中包括：

### CurdRepository

> 接口提供了最基本的对实体类的添删改查操作

### PagingAndSortingRepository

> 该接口提供了分页与排序功能

> **Pageable对象**

封装分页信息对象，实现类为Pagequest，包括:

> **Sort对象**

封装排序信息

> **Page对象**

封装查询分页信息对象

。。。

## JpaRepository

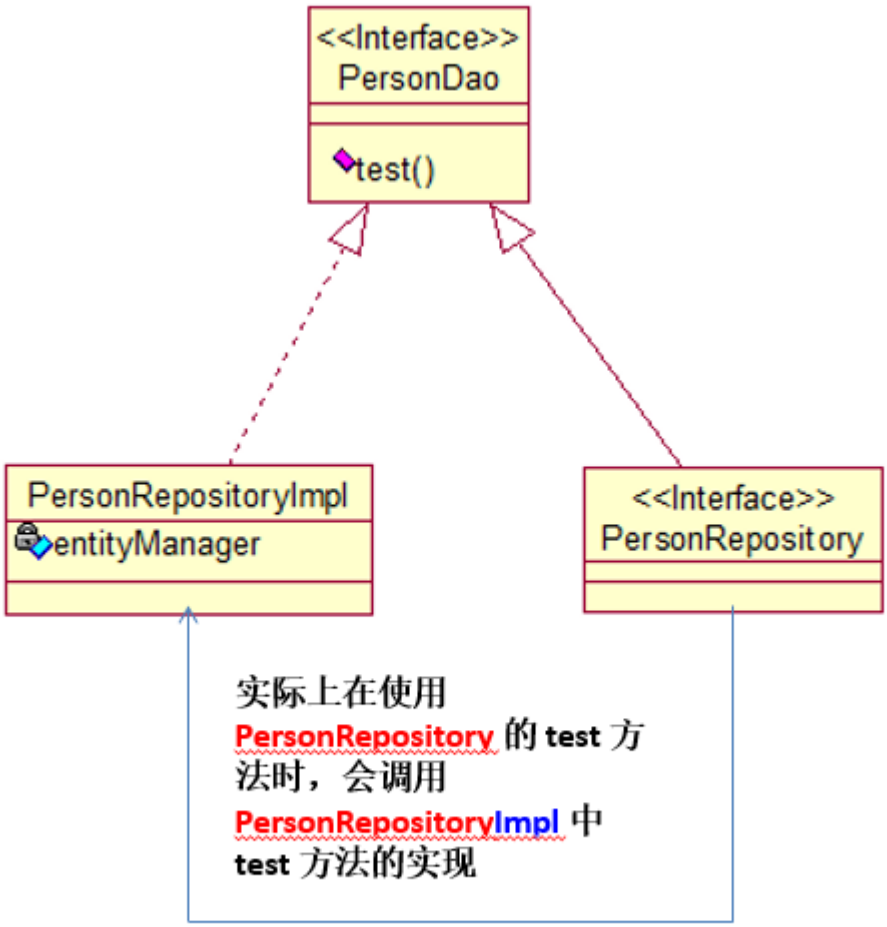
> 该接口提供了JPA的相关功能

## 自定义Repository

> 可根据需求自己实现查询方法

> 步骤

1. 定义一个接口: 声明要添加的, 并自实现的方法
2. 提供该接口的实现类: 类名需在要声明的 Repository 后添加 Impl, 并实现方法
3. 声明 Repository 接口, 并继承 1) 声明的接口使用.
4. 注意: 默认情况下, Spring Data 会在 base-package 中查找 "接口名Impl" 作为实现类. 也可以通过repository-impl-postfix　声明后缀.




## JpaSpecificationExecutor

不属于Repository体系，实现一组JPA Criteria 查询相关的方法。

> Specification对象

封装JPA Criteria查询的查询条件

```
        Specification<Student> studentSpecification = new Specification<Student>() {
            /**
             * @param *root: 代表查询的实体类.
             * @param criteriaQuery: 可以从中可到 Root 对象，即告知 JPA Criteria 查询要查询哪
一个实体类. 还可以
             * 来添加查询条件，还可以结合 EntityManager 对象得到最终查询的 TypedQuery 对象.
             * @param *criteriaBuilder 对象. 用于创建 Criteria 相关对象的工厂. 当然可以从中获
取到 Predicate 对象
             * @return: *Predicate 类型，代表一个查询条件.
             */
            public Predicate toPredicate(Root<Student> root, CriteriaQuery<?>
criteriaQuery, CriteriaBuilder criteriaBuilder) {
                Path<Object> teacher_id_path = root.get("teacher");
```

```
                Predicate predicate = criteriaBuilder.equal(teacher_id_path,1);
                return predicate;
            }
        };
```

# 删改

**@Quer**y 与 **@Modifying**这两个annotation一起声明，可定义个性化更新操作，注意JPQL不支持INSERT

示例如下：

```
@Modifying
@Query("UPDATE Customer c set c.customerName=?1")
int updateCustomerName(String cn)
```

**注意**

1. 方法的返回值应该是 int，表示更新语句所影响的行数
2. 在调用的地方必须加事务，没有事务不能正常执行

# 事务

- Spring Data 提供了默认的事务处理方式，即所有的查询均声明为只读事务。
- 对于自定义的方法，如需改变 Spring Data 提供的事务默认方式，可以在方法上注解 @Transactional 声明
- 进行多个 Repository 操作时，也应该使它们在同一个事务中处理，按照分层架构的思想，这部分属于业务逻辑层，因此，需要在 Service 层实现对多个 Repository 的调用，并在相应的方法上声明事务。

# HelloSpringData

使用 SpringData JPA 进行持久层开发需要的三个步骤：

1. **在 Spring的配置文件中配置SpringData**

   ```
   <jpa:repositories base-package="dao"
                     entity-manager-factory-ref="entityManagerFactory"
                     transaction-manager-ref="transactionManager"/>
   ```

2. **声明持久层的接口，该接口继承 Repository**

Repository 是一个标记型接口（类似标识对象序列化的Serializable），它不包含任何方法，如必要，SpringData 可实现 Repository其他子接口，其中定义了一些常用的增删改查，以及分页相关的方法。

```
public interface StudentRepositoryDao extends Repository<Student, Integer> {
```

3. **在接口中声明需要的方法**

   Spring Data 将根据给定的策略来为其生成实现代码。

```
public interface StudentRepositoryDao extends Repository<Student, Integer> {
    Student findById(Integer id);
}
```