# Spring Data JPA 持久层开发步骤

## 配置Spring 整合JPA

* 主要是引入依赖关系：spring、jpa、数据库驱动、连接池、日志框架

## Spring配置文件中配置Spring Data

1. 配置数据源:c3p0;
2. 配置 JPA 的 EntityManagerFactory;
3. 配置事务管理器;
4. 配置支持注解的事务;
5. 配置 SpringData.



## 声明持久层的接口,集成Repository

* **注释：**

**Repository 是一个标记型接口，它不包含任何方法，如必要，Spring Data 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。**

## 在接口中声明需要的方法

* **注释：**

**Spring Data 将根据给定的策略（具体策略稍后讲解）来为其生成实现代码。**

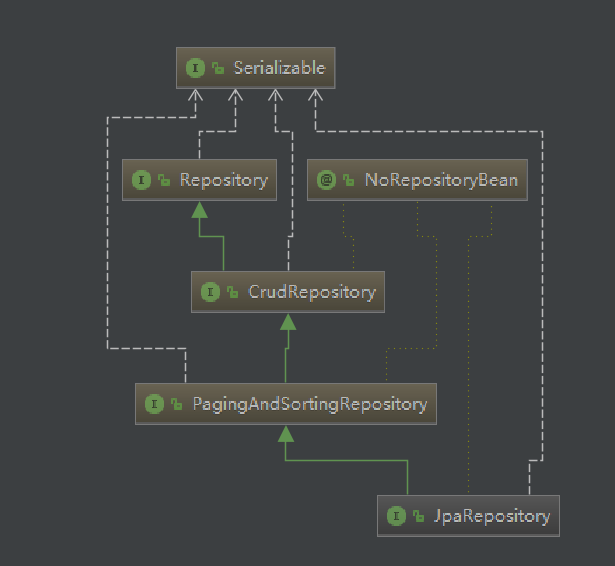
# Repository接口

* **注释：**

1. Repository是一个空接口，即是一个标记接口；
2. 若我们定义的接口**继承了Repository**，并且配置了**扫描 Repository Bean 所在的 package**，则该接口会被IOC容器识别为一个Repository Bean,纳入到IOC容器中，（并且容器会将该接口生成一个代理类）。进而可以在该接口中定义满足一定规范的方法。
3. 实际上，@RepositoryDefinition 注解，并为其指定 domainClass 和 idClass 属性，与**继承了Repository**是等价的。

# Repository子接口

## 子接口关系图：



## CrudRepository

* **注释：**

CrudRepository： 继承 Repository，实现了一组 CRUD 相关的方法。

## PagingAndSortingRepository

* **注释：**

PagingAndSortingRepository： 继承 CrudRepository，实现了一组分页排序相关的方法

## JpaRepository

* **注释：**

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

## 自定义的 XxxxRepository

* **注释：**

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

* **注释：**

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

# SpringData 方法定义规范

## 方法定义规范说明：

1. **不是随便声明，需要符合一定规范，否则编译错误；**
2. **查询方法以 find | read | get 开头；**
3. **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。**
4. **支持属性级联查询，若当前类有符合条件的属性，优先使用，而不适用级联属性，若想使用级联属性，则属性之间使用“\_”进行连接。**

## 举例说明：

1. **WHERE lastName LIKE %? AND id < ?**

|  |
| --- |
| List<Person> getByLastNameEndingWithAndIdLessThan  (String lastName, Integer id); |

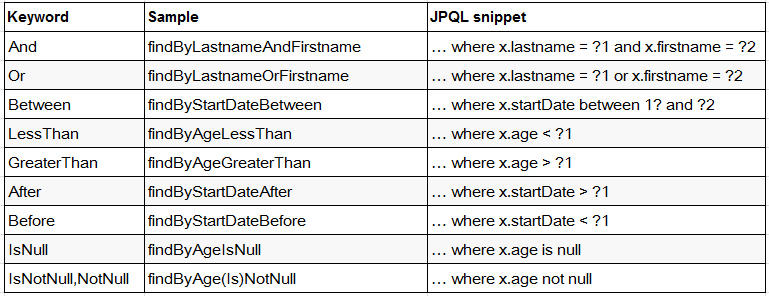
1. **级联查询：地址ID > ?**

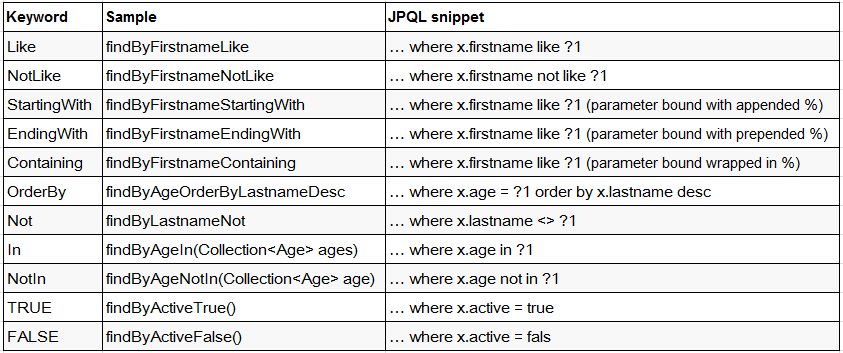
|  |
| --- |
| List<Person> getByAddressIdGreaterThan(Integer id); |

1. **支持属性级联查询，若当前类有符合条件的属性，优先使用，而不适用级联属性，若想使用级联属性，则属性之间使用“\_”进行连接。**

|  |
| --- |
| List<Person> getByAddress\_IdGreaterThan(Integer id); |

## 支持的关键字：

****

****

# @Query注解

## 使用背景：

**使用方法定义不能满足复杂的查询，因此需要@Query(jpql)进行复杂的查询。**

## @Query传递参数的方式

1. **使用占位符**：

|  |
| --- |
| @Query("SELECT p FROM Person WHERE p.lastName = **?1** AND p.email = **?2**")  List<Person> testQueryAnnotationParams1(String lastName, String email); |

1. **命名参数方式：**

|  |
| --- |
| @Query("SELECT p FROM Person WHERE  p.lastName = **:lastName** AND p.email = **:email**")  List<Person>testQueryAnnotationParams2(**@Param("lastName")**  String lastName, **@Param("email")** String email); |

1. **SpringData LIKE查询时，建议在占位符上添加%**

|  |
| --- |
| @Query("SELECT p FROM Person WHERE  p.lastName LIKE **%?1%** OR p.email LIKE **%?2%**")  List<Person>testQueryLike(String lastName, String email); |

## 子查询:

1. **查询 id 值最大的Person(需要子查询)**

|  |
| --- |
| **@Query("SELECT p FROM Person WHERE p.id = (SELECT max(p2.id) FROM Person p2")**  **Person getMaxIdPerson();** |

## 原生sql查询

1. 设置nativeQuery=true

|  |
| --- |
| @Query(value="select count(id) from jpa\_person  nativeQuery=true")  long getTotalCount(); |

# @Modifying 注解和事务

* **注释：**

1. **在springData中一般通过save方法来做insert、update,其中jpa会根据对象的主键在判断是insert OR update,但是spring data jpa 是可以进行update\delete操作的。但是需要添加@Modifying 和事物才能通过@Query进行update操作，(jpql不支持insert)例如**
2. **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**

|  |
| --- |
| **Repository:**  @Modifying  @Query("UPDATE Person p SET p.email = :email WHERE id = :id")  void updatePersonEmail(@Param("id") Integer id,  @Param("email") String email);  **Service:**  @Transactional  public void updatePersonEmail(String email, Integer id){  personRepsotory.updatePersonEmail(id, email);  } |

# CrudRepository 接口

* **CrudRepository 接口提供了最基本的对实体类的添删改查操作**

1. T save(T entity);//保存单个实体
2. Iterable<T> save(Iterable<? extends T> entities);//保存集合
3. T findOne(ID id);//根据id查找实体
4. boolean exists(ID id);//根据id判断实体是否存在
5. Iterable<T> findAll();//查询所有实体,不用或慎用!
6. long count();//查询实体数量
7. void delete(ID id);//根据Id删除实体
8. void delete(T entity);//删除一个实体
9. void delete(Iterable<? extends T> entities);//删除一个实体的集合
10. void deleteAll();//删除所有实体,不用或慎用!

# PagingAndSortingRepository

## 分页对象

* **Pageable是一个接口，具体的实现类是PageRequest，具体属性如下：**

private final int pageNumber; //页码,是从0开始

private final int size; //页面大小

private final Sort sort; //排序

* **Page是一个接口，封装了分页返回的查询信息，具体实现PageImpl**

**page.getTotalElements(); //总记录数**

**page.getNumber(); //当前第几页，是从0开始，应该 + 1**

**page.getTotalPages(); //总页数**

**page.getContent(); //当前页面的List**

**page.getNumberOfElements(); //当前页面的记录数**

* **Sort 封装了排序信息**

**其中Order封装了排序信息：升序、降序,属性**

|  |
| --- |
| Order order1 = new Order(Direction.DESC, "id");  Order order2 = new Order(Direction.ASC, "email");  Sort sort = new Sort(order1, order2);  int pageNo = 6 – 1;  int pageSize = 5;  Pageable pageable = new PageRequest(pageNo, pageSize, order);  Page<Person> page = personRepository.findAll(pageable); |

## API接口

1. Iterable<T> findAll(Sort sort);
2. Page<T> findAll(Pageable pageable);

# JpaRepository

1. List<T> findAll(); //查找所有实体
2. List<T> findAll(Sort sort); //排序、查找所有实体
3. List<T> save(Iterable<? extends T> entities);//保存集合
4. void flush();//执行缓存与数据库同步
5. T saveAndFlush(T entity);//强制执行持久化
6. void deleteInBatch(Iterable<T> entities);//删除一个实体集合

# JpaSpecificationExecutor

## 出现背景：

因为PagingAndSortingRepository接口不能查询带条件的分页，所以才出现了JpaSpecificationExecutor

## 举例说明：

1. **调用JpaSpecificationExecutor的Page<T>**

**findAll(Specification spec, Pageable pageable);**

1. **Specification：封装了JPA Criteria 查询的查询条件**
2. **Pageable:封装了请求分页的信息：例如pageNo,pageSize,Sort**
3. **代码如下：id > 5 的第3页数据**

**参数解释：**

* **Root<Person> root：代表查询类，即实体类，他能提供实体类中属性的导航路径Path**
* **CriteriaQuery<?> query:可以从中得到Root对象，即可告知JPA Criteria查询哪一个实体类，还可以添加查询条件，还可以结合EntityManager对象得到最终查询的TypedQuery对象；**
* **CriteraBuilder cb：用于创建Criteria相关对象的工厂，当然可以从中获取到Predicate对象，通过root提供固定属性的Path**

|  |
| --- |
| @Test  public void testJpnSpecificationExecutor() {  int pageNo = 3-1;  int pageSize = 5;  Pageable pageable = new PageRequest(pageNo, pageSize);  //通常使用Specification的匿名内部类  Specification<Person> specification = new Specification<>{  @Override  public Predicate toPredicate(Root<Person> root,  CriteriaQuery<?> query, CriteraBuilder cb) {  Path path = root.get("id");  Predicate predicate = cb.gt(path, 5);  return predicate;  }  }  Page<Person> page = personRepository.findAll(specification,  pageable);  } |

## 方法列表

* 1. T findOne(Specification<T> spec);
  2. List<T> findAll(Specification<T> spec);
  3. Page<T> findAll(Specification<T> spec, Pageable pageable);
  4. List<T> findAll(Specification<T> spec, Sort sort);
  5. long count(Specification<T> spec);

# 自定义 Repository 方法

1. 为某一个 Repository 上添加自定义方法（复杂报表查询）

步骤：

* 定义一个接口: 声明要添加的, 并自实现的方法
* 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
* 声明 Repository 接口, 并继承 1) 声明的接口

**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.**

**Example:**

|  |
| --- |
| **自定义接口：**  public interface PersonDao {  void test();  }  **自定义接口实现：**  public class PersonRepsotoryImpl implements PersonDao {    @PersistenceContext  private EntityManager entityManager;    @Override  public void test() {  Person person = entityManager.find(Person.class, 11);  System.out.println("-->" + person);  }  }  **PersonRepsotory集成PersonDao：**  public interface PersonRepsotory extends  JpaRepository<Person, Integer>,  JpaSpecificationExecutor<Person>, PersonDao {} |

1. 为所有的 Repository 都添加自实现的方法（不常用）