**Spring data jpa**

## 一、Spring Data JPA简介

**1 Spring基于Hibernate开发的一个JPA框架。**如果用过Hibernate或者MyBatis的话，就会知道对象关系映射（ORM）框架有多么方便。但是Spring Data JPA框架功能更进一步，为我们做了 一个数据持久层框架几乎能做的任何事情。下面来逐步介绍它的强大功能。是Spring在ORM框架，以及JPA规范的基础上，封装的一套JPA应用框架，并提供了一整套的数据访问层解决方案。

**2 Spring Data是一个用于简化数据库访问，并支持云服务的开源框架。**其主要目标是使得对数据的访问变得方便快捷，并支持map-reduce框架和云计算数据服务。

**3 Spring Data 包含多个子项目：**

Commons - 提供共享的基础框架，适合各个子项目使用，支持跨数据库持久化

JPA - 简化创建 JPA 数据访问层和跨存储的持久层功能

Hadoop - 基于 Spring 的 Hadoop 作业配置和一个 POJO 编程模型的 MapReduce 作业

Key-Value - 集成了 Redis 和 Riak ，提供多个常用场景下的简单封装

Document - 集成文档数据库：CouchDB 和 MongoDB 并提供基本的配置映射和资料库支持

Graph - 集成 Neo4j 提供强大的基于 POJO 的编程模型

Graph Roo AddOn - Roo support for Neo4j

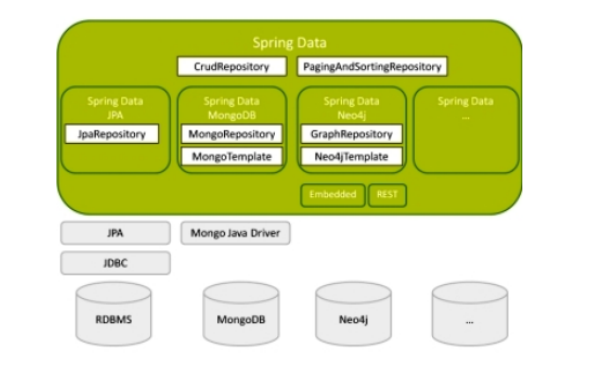
JDBC Extensions - 支持 Oracle RAD、高级队列和高级数据类型

Mapping - 基于 Grails 的提供对象映射框架，支持不同的数据库

Examples - 示例程序、文档和图数据库

Guidance - 高级文档

**4 Spring Data 项目的目的是为了简化构建基于 Spring 框架应用的数据访问计数，包括关系数据库、非关系数据库、Map-Reduce 框架、云数据服务的访问支持。统一和简化对各类型持久化存储， 而不拘泥于是关系型数据库还是NoSQL 数据存储。**



**5 Spring社区的一个顶级工程**，主要用于简化数据（关系型&非关系型）访问，如果我们使用Spring Data来开发程序的话，那么可以省去很多低级别的数据访问操作，如编写数据查询语句、DAO类等，我们仅需要编写一些抽象接口并定义相关操作即可，Spring会在运行期间的时候创建代理实例来实现我们接口中定义的操作。

**Spring Data JPA**

**Spring Data Commons**

**Spring Data MongoDB**

**Spring Data Redis**

**Spring Data Solr**

**Spring Data Gemfire**

**Spring Data REST**

**Spring Data Neo4j**

**6 Spring data JPA的功能**

**Repository：**仅仅是一个标识，表明任何继承它的均为仓库接口类，方便Spring自动扫描识别

**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

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

**JpaSpecificationExecutor：**比较特殊，不属于Repository体系，实现一组JPA Criteria查询相关的方法。

**7 maven依赖**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

**8 yml文件 配置mysql,jpa等**

server:

port: 8080

context-path: /

helloWorld: spring Boot\u5927\u7237\u4F60\u597D

msyql:

jdbcName: com.mysql.jdbc.Driver

dbUrl: jdbc:mysql://localhost:3306/wj33

userName: root

password:

spring:

datasource:

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost:3306/wj33?useUnicode=true&characterEncoding=utf-8&useSSL=false

username: root

password:

jpa:

hibernate.ddl-auto: update

show-sql: true

database-platform: com.lunwen.wangjie.config.MySQL5DialectUTF8

thymeleaf:

cache: false

**9 实体类**

实体类的注解 会根据注解自动生成表字段，

@Table指定表名，不填，默认为类名

@Column指定字段名，不填，默认为属性名

其他注解指定字段的规则

import javax.persistence.\*;

@Entity

@Table(name = "user")

public class User

{

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

/\*\* 姓名 \*/

@Column(name = "name")

private String name;

/\*\* 密码 \*/

@Column(name = "password")

private String password;

/\*\* email \*/

@Column(name = "email")

private String email;

/\*\* level 用于判断学生0 教师1 管理员2\*/

@Column(name = "level")

private int level;

/\*\* 学号 工号 \*/

@Column(name = "number", unique = true)

private String number;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public int getLevel() {

return level;

}

public void setLevel(int level) {

this.level = level;

}

public String getNumber() {

return number;

}

public void setNumber(String number) {

this.number = number;

}

}

**10 数据访问层**

操作数据访问的接口，支持三种查询，hql, sql, 方法名语义分析。

下列代码是通过hql查询，也就是通过类名，属性名

import com.lunwen.wangjie.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

public interface UserDao extends JpaRepository<User, Long>{

@Query("from User u where u.name=:name")

User findUser(@Param("name") String name);

@Query("from User u where u.email=:email")

User findUserByEmail(@Param("email") String email);

@Query("from User u where u.number=:number")

User findUserByNumber(@Param("number") String number);

User findUserByNumber(String numer);

}

也可能过原生的sql查询@Query(“原生sql”)

也可通过方法名拼接如下

findAllByWno，会语义分析为查找所有的当前类，通过wno字段，从数据库中去捞数据，一般Idea会自动告诉你如何拼接可行的方法名。

还可以继承分页的数据访问Repository,现在继承的是JpaRepository，里面指定类名，就会关联的数据库中的表。

import com.lunwen.wangjie.model.StudentWork;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface StudentWorkDao extends JpaRepository<StudentWork, Long>{

List<StudentWork> findAllByWno(Long wno);

List<StudentWork> findAllBySno(String sno);

StudentWork findStudentWorkBySnoAndWno(String sno, Long wno);

}

**11 急切和延迟加载**

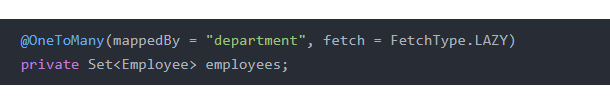
1 JPA中最重要的概念是为了使数据库的副本在高速缓冲存储器中。虽然有一个数据库事务，但JPA首先创建一个重复的数据集，只有当它使用实体管理提交，所做的更改影响到数据库中。

2 从数据库中获取记录有两种方式

（1）预先抓取，相关子对象获取一个特定的记录自动上传

（2）延迟加载，在延迟装载，涉及的对象不会自动上传，除非你特别要求他们。

@Basic、 @OneToMany、@ManyToOne、@OneToOne、@ManyToMany等所有的注解都有一个被称为fetch的可选参数，如果该参数 被设置为FetchType.LAZY的话，那么对于JPA提供程序来说，它会被解释成一个提示，示意可以延迟该域的加载直至其第一次被访问：



## 二、高级知识

**1 级联**

**1 cascade:**定义类和类之间的级联关系。级联关系定义对当前对象的操作将波及到关联类的对象，而且这个种关系是递归调用的。如School与Student有级联删除关系，那么删除School时将同时删除关联的Student对象。

如果Student还与其他关联，会递归删除下去。

**2 CascadeType.PERSIST** 级联持久化（保存）操作（持久保存拥有方实体时，也会持久保存该实体的所有相关数据。）

**3 CascadeType.REMOVE Cascade remove operation，**级联删除操作。 删除当前实体时，与它有映射关系的实体也会跟着被删除。

**4 CascadeType.MERGE Cascade merge operation，**级联更新（合并）操作。当Student中的数据改变，会相应地更新Course中的数据

**5 CascadeType.DETACH Cascade detach operation，**级联脱管/游离操作。如果你要删除一个实体，但是它有外键无法删除，你就需要这个级联权限了。它会撤销所有相关的外键关联。

**6 CascadeType.REFRESH Cascade refresh operation，**级联刷新操作。假设场有一个订单,订单里面关联了许多商品,这个订单可以被很多人操作,那么这个时候A对此订单和关联的商品进行了修改,与此同时,B也进行了相同的操作,但是B先一步比A保存了数据,那么当A保存数据的时候,就需要先刷新订单信息及关联的商品信息后,再将订单及商品保存。(来自良心会痛的评论) CascadeType.ALL

**7 Cascade all operations，**清晰明确，拥有以上所有级联操作权限。

**2 JPA实体关系**

实体关系是指实体与实体之间的关系，从方向上分为单向关联和双向关联，从实体数量上分为一对一，一对多，多对多等，对于任何两个实体，都要从这个两个方面区分它们之间的关系。单向关联是一个实体中引用了另外一个实体，也即通过一个实体可以获取另一个实体对象的引用，双向关联是两个实体之间可以互相获取对方对象的引用。

**1 一对一 @OneToOne( 学生证和身份证是一对一的关系。)**

(1)单向关联

假设在学生实体中可以获取身份证对象的引用；反之，不能再身份证实体中获取学生对象的引用，则学生和身份证是一对一的单向关联关系

这时候只需要在学生实体中的IDCrad\_id加上@JoinColumn(name = “外键表字段名”)，以及@OneToOne(cascade=CascadeType.All)

@OneToOne 只能确定实体与实体之间的关系是一对一的关系，不能指定数据库表中的保存的关联字段，所以要结合@JoinColumn来指定关联字段

在默认情况下，关联实体(IDCard)的主键一般是用来做外键的，但如果此时不想将关联实体的主键作为外键，需要设置@JoinColumn的referencedColumnName属性

(2)双向关联

需要在接受方的字段定义一个@OneToOne(mappedBy = “主动方一对一的属性名”)

(3)主键关联

既让两个实体对象具有相同的主键值，以表明它们之间的一一对应关系；而数据库不会有额外的字段来维护它们之间的关系，仅通过表的主键来关联，主键的值需要程序来显示维护。

只要在@OneToOne的下面添加一个@PrimaryKeyJoinColumn

(4)总结

@确定实体与实体之间的关系，如果是一对一关系 则使用@OneToOne

@确定表的结构的设计

如果是外键关联，在关系维护段考虑默认的实体关系映射或@JoinColumn

如果表位于不同的数据中，可以采用主键关联，使用@PrimaryKeyJoinColumn

@确定实体关系的方向

单向关系 在保存关联关系的实体中，使用@JoinColumn

双向关系 则在保存关联关系（也即存在外键）的实体中，要配合@JoinColumn；在没有保存关联关系的实体中，需使用mappedBy属性明确所关联的实体

mappedBy用在双向关联中，mappedBy所在的实体为关系被维护端，而另一个实体为关系维护端（也即保存关联关系的一端）

**2 一对多**

(1)单向关联

使用@ManyToOne 比如一个学校可能有多个学生 ，那就可以在学生的schoolId字段加上@JoinColumn(name=”school\_id”) @ManyToOne

(2)双向关联

在学校的students字段加一个 @OneToMany(mappedBy=”school”, cascade=CascadeType.ALL) 学生表中的SchoolId字段@JoinColumn(name = “”) @ManyToOne

option：指定级联方是否可以为空，默认为true,允许外键字段为空，若将其设置为false,则双方必须存在，也即外键字段不能为空

(3)表关联（双向）

在一对多或多对一的关联关系中，除了使用默认的外键关联外；还可以使用表关联，多的一方可以作为关系表的主键（唯一性约束）,而一的一方可以作为关系表另一个字段。

在多的一方作为关系表的主键 name : 中间表 joinColumns:中间表中，指向关系维护端的外键； inverseJoinColumns:与joinColumns相似，指向中间表中被维护端的外键

java

@ManyToOne

@JoinTable(name="SCHOOL\_STUDENT",

joinColumns={

@JoinColumn(name="student\_id",referencedColumnName="id") //自方

},

inverseJoinColumns={

@JoinColumn(name="school\_id",referencedColumnName="id") //一方

})

在一的一方 @OneToMany(mappedBy=”school”)

(4)总结

@确定实体与实体之间的关系，如果是一对多的关系，则使用@OneToMany；如果是多对一的关系使用@ManyToOne

@确认表结构的设计

如果是外键关联，在关系维护端考虑默认的实体关系映射或配合JoinColumn

如果是表关联，则在关系维护端使用@JoinTable

@确定实体关系的方向

单向关联 一般情况下，多的一方为关系维护端，在保存关联关系实体中使用@JoinColumn或@JoinTable

若为双向关联，则在关系维护端，配合@JoinColumn或@JoinTable；在关系呗维护端，要使用mappedBy属性明确所关联的实体

**3 多对多**

(1)单向关联

老师和学生就是一个多对多关系 ,在老师实体类中维护

java

@ManyToMany(fetch=FetchType.LAZY)

@JoinTable(name="TEACHER\_STUDENT",

joinColumns={@JoinColumn(name="teacher\_id")},

inverseJoinColumns={@JoinColumn(name="student\_id")})

(2)双向关联

在老师中如上维护，在学生中处理接受方 @ManyToMany(mappedBy=”students”,fetch=FetchType.LAZY)

(3)总结

单向关联在维护端使用@JoinTable

双向关联在维护端使用@JoinTable 在关系被维护端，要使用mappedBy属性明确所关联

FetchType.EAGER:代表立即加载；

FetchType.LAZY:代表延迟加载。

**3 数据的级联抓取(Fetch)**

**1） 基础知识**

级联抓取(Fetch)是JPA提供的实体键的关联属性，用于执行实体的读操作，同时对其关联的其他实体进行操作

@如果不配置fetch属性，则缺省值为

@如果对端为”一”,即@OneToOne或@ManyToOne映射的属性，缺省为FetchType.EAGER，即缺省会将关联数据抓取出来。

@如果对端为”多”,即@OneToMany或@ManyToMany映射的属性，缺省为FetchType.LAZY,即缺省不会将关联的数据抓取出来。

使用 JPA 级联抓取数据时，可通过指定 EntityGraph 配置连接关系 @EntityGraph(attributePaths = “extra”) ，EntityGraph 已经将实体本身的 fetch 属性覆盖了，即使配置了 FetchType.LAZY，只要在查询时配置了 EntityGraph，关联的对象将以最高效的方式查询出来。

如果 Fetch 属性配置在实体上，那么所有对实体的查询操作都将受其影响，无论这些关联数据是否需要都会被查出来。而使用 EntityGraph，配置到数据访问方法上，则可以很精确的配置究竟需要哪些关联数据，粒度更精准。

为了避免无谓的效率浪费，我们建议：所有的 @OneToOne 和 @ManyToOne 关联实体，均应配置 fetch = FetchType.LAZY，避免缺省将关联实体查询出来。当确实需要使用关联实体时，在数据访问方法中使用 EntityGraph 配置。

@Transactional 注解；同时由于要执行数据保存操作，故需要增加 @Commit 注解，使得事物在方法退出后提交。如果只写 @Transactional，不写 @Commit，则方法退出后事物缺省回滚

**2 ）spring框架对Jpa提供了以下的支持**

它使得 JPA 配置变得更加灵活。JPA 规范要求，配置文件必须命名为 persistence.xml，并存在于类路径下的 META-INF 目录中。该文件通常包含了初始化 JPA 引擎所需的全部信息。Spring 提供的 LocalContainerEntityManagerFactoryBean 提供了非常灵活的配置，persistence.xml 中的信息都可以在此以属性注入的方式提供

其次，Spring实现了部分在EJB容器 下才具有的功能

Spring将EntityManager的创建和销毁，事务管理等代码抽取出来，并由其统一管理

**3 ）Spring Data Jpa框架主要针对的就是Spring唯一没有简化到的业务逻辑代码**，不用实现持久层的业务逻辑，唯一需要做的是声明持久层的接口，其他都交托给spring data Jpa完成。

**4） 让持久层接口DAO,集成Repository接口**

该接口使用了泛型，需要为其提供两个类型：第一个为该接口处理的域对象类型，第二个为该域对象的主键类型。如下：

public interface UserDao extends Repository<AccountInfo, Long> {

public AccountInfo save(AccountInfo accountInfo);

}

不需要使用UserDao的实现类，框架会为我们完成业务逻辑，在spring配置文件中启用扫描并创建代理的功能

<-- 需要在 <beans> 标签中增加对jpa命名空间的引用 -->

<jpa:repositories base-package="footmark.springdata.jpa.dao"

entity-manager-factory-ref="entityManagerFactory"

transaction-manager-ref="transactionManager" />

**5） spring data jpa进行持久层开发大致需要的三个步骤**

声明持久层的接口，该接口继承Repository,Repository是一个标记型接口，它不包含任何方法，当然如果有需要，Spring Data 也提供了若干的Repository的子接口，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业务方法，spring data 将根据给定的策略来为其生成实现代码。

在spring配置文件中增加一行声明，让spring为声明的接口创建代理对象。配置了后，spring初始化容器时会扫描base-package指定的包目录及其子目录，为继承Repository或其子接口的接口创建代理对象，并将代理对象注册为Spring bean,业务层便可以通过spring自动封装的特性来直接使用该对象。

此外还提供了一些属性和子标签，便于做更细粒度的控制，可以在内部使用、来过滤掉一些不希望被扫描的接口

**6） 如果持久层接口较多，且每一个接口都需要声明相似的增删改查**，可以继承CrudRepository接口，会自动为域对象创建增删改查方法，供业务层使用，可能暴露了不希望暴露给业务层的方法

**7） 分页查询和排序是持久层常用的功能，spring data**

**8） 为此提供了PagingAndSortingRepository接口**，它继承自CrudRepository接口，在CrudRepository基础上新增了两个与分页有关的方法。但是，我们很少会将自定义的持久层接口直接继承自PagingAndSortingRepository，而是在继承Repository或CrudRepository,在自己声明的方法的参数列表加一个Pageable或Sort类型的参数，用于指定分页或排序信息即可，这比直接使用PagingAndSortingRepository提供了更大的灵活性。

**9） JpaRepository是继承自PagingAndSortingRepository的针对JPA提供接口**，它在父及接口的基础上提供了其他一些方法，比如flush(),saveAndFlush,deleteInBatch().

4 **查询方法**

**1）** 框架会对方法名进行解析，会把方法名多余的前辍截取掉，比如find,findBy,read,readBy,get,getBy,然后对剩下部分进行解析。并且如果方法的最后一个参数是Sort或者Pagealbe类型，也会提取相关信息，以便按规则进行排序或者分页查询。

2**）** 在创建查询时，我们通过在方法名中使用属性名称来表达，比如findByUserAddressZip(). 首先剔除findBy,然后对剩下属性进行解析，详细规则如下（此处假设该方法针对的域对象为AccountInfo类型）

1. 先判断userAddressZip(根据POJO规范,首字母变为小写，下同),是否为AccountInfo的一个属性，如果是，则表示根据该属性进行查询；如果没有该属性，继续第二步；
2. 从右往左边截取第一个大写字母开头的字符串（此处是Zip),然后检查剩下的字符串是否为AccountInfo的一个属性，如果是，则表示根据该属性进行查询，如果没有该属性，则重复第二步，继续从右往左进行字符串截取，最后假设user为AccountInfo的一个属性

（3）接着处理剩下部分（AddressZip)，先判断user所对于的类型是否有addressZip属性，如果有，则表示该方法最终是根据”AccountInfo.user.addressZip”的取值进行查询的，否则继续按照步骤2的规则从右往左边截取，最终表示更加”AccountInfo.user.address.zip”的值进行查询。否则继续按照步骤2的规则从右往截取；最后表示根据”AccountInfo.user.addrsss.zip”的值进行查询。

3 **）**在查询时通常需要同时根据多个属性进行查询，且查询的条件也各式各样（大于某个值，在某个范围等等）,spring data jpa为此提供了一些表达条件查询的关键字

**And —** 等价于 SQL 中的 and 关键字，比如 findByUsernameAndPassword(String user, Striang pwd)；

**Or —** 等价于 SQL 中的 or 关键字，比如 findByUsernameOrAddress(String user, String addr)；

**Between —** 等价于 SQL 中的 between 关键字，比如 findBySalaryBetween(int max, int min)；

**LessThan —** 等价于 SQL 中的 “<”，比如 findBySalaryLessThan(int max)；

**GreaterThan —** 等价于 SQL 中的”>”，比如 findBySalaryGreaterThan(int min)；

**IsNull —** 等价于 SQL 中的 “is null”，比如 findByUsernameIsNull()；

**IsNotNull —** 等价于 SQL 中的 “is not null”，比如 findByUsernameIsNotNull()；

**NotNull —** 与 IsNotNull 等价；

**Like —** 等价于 SQL 中的 “like”，比如 findByUsernameLike(String user)；

**NotLike —** 等价于 SQL 中的 “not like”，比如 findByUsernameNotLike(String user)；

**OrderBy —** 等价于 SQL 中的 “order by”，比如 findByUsernameOrderBySalaryAsc(String user)；

**Not —** 等价于 SQL 中的 “！ =”，比如 findByUsernameNot(String user)；

**In —** 等价于 SQL 中的 “in”，比如 findByUsernameIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；

**NotIn —** 等价于 SQL 中的 “not in”，比如 findByUsernameNotIn(Collection userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；

4 @Query创建查询

（1）@Query注解的使用非常简单，只需要在声明的方法上面标注该注解，同时提供一个JPQL查询语句即可如下所示

public interface UserDao extends Repository<AccountInfo, Long> {

@Query("select a from AccountInfo a where a.accountId = ?1")

public AccountInfo findByAccountId(Long accountId);

@Query("select a from AccountInfo a where a.balance >?1")

public Page<AccountInfo> findByBalanceGreaterThan(Integer balance, Pageable pageable);

}

（2）创建JP QL时用明明参数来代替位置编号，@Query也对此提供了支持。JP QL中通过”:变量”的格式来制定参数，同时在方法的参数前面使用@Param将方法参数与JP QL中的命名参数对应。

public interface UserDao extends Repository<AccountInfo, Long> {

@Query("from AccountInfo a where a where a.ccountId = :id")

public AccountInfo findByAccountId(@Param("id")Long accountId);

@Query("from AccountInfo a where a.banlace > :balance")

public Page<AccountInfo> findByBalanceGreaterThan(@Param("balance")Integer balance,Pageable pageable);

}

（3）此外可以使用@Query来执行一个更新操作，为此我们需要在使用@Query的同时，用@Modifying来将该操作标识符标注就好了

@Modifying

@Query("updata AccountInfo a set a.salary = ?1 where a.salary < ?2")

public int increaseSalary(int after,int before);

（4）通过JPA命名查询语句创建查询

命名查询是JPA提供的一种将查询语句从方法体重独立出来，以供多个方法共用的功能

（5）JpaRepository 接口特色

将一些查询方法返回类型由Iterable转换成了List

新增了保存或更细的方法

List<T> findAll();

List<T> findAll(Sort sort);

List<T> findAll(Iterable<ID> ids);

<S extends T> List<S> save(Iterable<S> entities);

void flush();

<S extends T> S saveAndFlush(S entity);

void deleteInBatch(Iterable<T> entities);

void deleteAllInBatch();

T getOne(ID id);

（6）JpaSpecificationExecutor特色：实现了带条件的查询，类似于Hiberante的cretira

T findOne(Specification<T>);

List<T> findAll(Specification<T>);

List<T> findAll(Specification<T>, Sort);

List<T> findAll(Specification<T>, Pageable);

long count(Specification<T>);

5 spring Data Jpa对事务的支持

**默认情况下，Spring Data JPA实现的方法都是使用事务的。针对查询类型的方法，其等价于@Transactional(readOnly=true);增删该查类型的方法，等价于@Transactional.除了查询的方法设为只读事务外，其他食物属性均采用默认值**。

## 三、 总结

**1 默认操作**

findAll()；findOne(1l)；save(user); delete(user); count(); exists(1l);

findXXBy,readAXXBy,queryXXBy,countXXBy, getXXBy后面跟属性名称

deleteById；countByUserName；findByEmailLike；findByUserNameOrderByEmailDesc；findByUserNameIgnoreCase

**2 组合条件查询**

findByLastnameAndFirstname … where x.lastname = ?1 and x.firstname = ?

indByLastnameOrFirstname … where x.lastname = ?1 or x.firstname = ?2

findByFirstnameIs,findByFirstnameEquals … where x.firstname = ?1

findByStartDateBetween … where x.startDate between ?1 and ?2

findByAgeLessThan … where x.age < ?1

**3 分页查询**

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

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

userRepository.findALL(pageable);

userRepository.findByUserName("testName", pageable);

**4 自定义查询**

@Transactional(timeout = 10) @Query("select u from User u where u.emailAddress = ?1") User findByEmailAddress(String emailAddress);

**5 多表查询**

首先需要定义一个结果集的接口类，查询的方法返回类型设置为新创建的接口

public interface HotelSummary {

City getCity();

String getName();}

@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);

Page<HotelSummary> hotels = this.hotelRepository.findByCity(new PageRequest(0, 10, Direction.ASC, "name"));

for(HotelSummary summay:hotels)

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

**6 多数据源支持**

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

Page<Menu> result = menuRepository.findAll(new Specification<Menu>()

{

@Override

public Predicate toPredicate(Root<Menu> root, CriteriaQuery<?> query, CriteriaBuilder cb)

{

List<Predicate> list = new ArrayList<Predicate>();

if (StringUtils.isNotEmpty(menuDto.getUrl())) //根据url模糊查询

{

list.add(cb.like(root.get("url").as(String.class), "%" + menuDto.getUrl() + "%"));

}

if (StringUtils.isNotEmpty(menuDto.getType())) //根据type精确查询

{

list.add(cb.equal(root.get("type").as(String.class), menuDto.getType()));

}

Predicate[] p = new Predicate[list.size()];

return cb.and(list.toArray(p));

}

}, pageable);

return result;

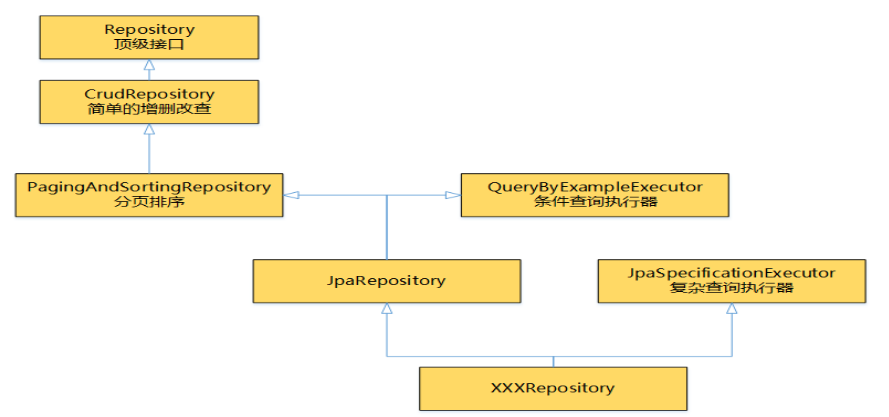
## 四、源码分析

spring-data-commons包是核心包。颗粒度划得很细，这主要也是为了责任分离。

Repository：空的接口，目的是为了统一所有Repository的类型，其接口类型使用了泛型，泛型参数中T代表实体类型，ID则是实体中id的类型

CrudRepository :简单的增删改查

PagingAndSortingRepository：分页排序



JpaSpecificationExecutor：其中Specification就是需要我们传进去的参数，它是一个接口

public interface Specification<T> {

Predicate toPredicate(Root<T> root, CriteriaQuery<?> query, CriteriaBuilder cb);}

public Predicate toPredicate(Root<Task> root, CriteriaQuery<?> query, CriteriaBuilder cb) {

Path<String> exp1 = root.get("taskName");

Path<Date> exp2 = root.get("createTime");

Predicate predicate = cb.and(cb.like(exp1, "%taskName%"),cb.lessThan(exp2, new Date()));

return cb.or(predicate,cb.equal(exp3, "kkk")); //1.混合条件查询

Join<Task,Project> join = root.join("project", JoinType.INNER);

Path<String> exp4 = join.get("projectName");

return cb.like(exp4, "%projectName%"); //2.多表查询

QueryByExampleExecutor：类似于mybatis的example简单 条件查询

UserExample example = new UserExample();

Criteria criteria = example.createCriteria();

criteria.andUsernameEqualTo("wyw");

example.setOrderByClause("username asc,email desc");

List<?>list = XxxMapper.selectByExample(example);

**类似Mybatis，使用动态代理和动态生成字节码技术，Spring boot为声明@Repository注解的接口创建代理实现类的对象，并将代理对象注册为 Spring Bean，业务层便可以通过 Spring 自动封装的特性来直接使用该对象。默认情况下，Spring Data JPA 实现的方法都是使用事务的。针对查询类型的方法，其等价于 @Transactional(readOnly=true)；增删改类型的方法，等价于@Transactional。可以在接口方法上使用 @Transactional 显式指定事务属性，该值覆盖 Spring Data JPA 提供的默认值。**